PYTHON

TKINTER

GUI PROJECTS

PRACTICAL EXERCISES FOR BEGINNERS

VAISHALI B. BHAGAT

PYTHON
Tkinter
gui
projects

practical exercises for BeginNers

VAISHALI B. BHAGAT

Cover design by Dr.Sushil B.Naik

ISBN :9798327318953

For

My Father

CONTENTS

- **Quadratic Equation Solver using the Factorisation Method**
- **Events and Holiday Calendar**

ACKNOWLEDGMENTS

I would like to express my deep and sincere gratitude to my parents for their love, prayers, caring and sacrifices for educating and preparing for my future.
I am extremely grateful to my husband Dr. Sushil Naik, without whose support I would not have been able to complete this book successfully.

Vaishali B.Bhagat

Simple Math Calculator

Project #1

Design simple math operation calculator using Python and Tkinter

Introduction:

Welcome to the Simple Math Operation Calculator project! In this program, we'll explore how to make a simple graphical user interface (GUI) using Python Tkinter. Our aim is to build an application that allow users to compute sums, differences, products, quotients, remainders, as well as find the biggest and smallest numbers effortlessly. To accomplish this, we'll utilize Tkinter, the most popular library for creating GUIs in Python. We'll design a user-friendly interface where users can enter two numbers via text boxes.

Additionally, we'll implement seven operation buttons, each representing a distinct mathematical operation: addition, subtraction, multiplication, division, modulo, finding the biggest number, and finding the smallest number.

Upon clicking any of these operation buttons, the calculator will compute the result and display it in a message box, providing users with immediate feedback on their calculation. Let's dive in and create our first Tkinter app!

Requirements:

Text Editor or IDE: You'll need a text editor to write the code. I've used the Thonny IDE for writing and executing this projects. However, you can use any Python IDE or text editor of your choice to run the program.

Python with Tkinter: Ensure that you have Python installed on your system. Tkinter is usually included with Python installations by default, so there's typically no separate installation needed. It's assumed that you have a basic understanding of Python

programming language concepts such as functions, variables, and control flow. We won't go deeply into these topics in this book.

Tkinter Widgets: We'll use three Tkinter widgets to design the GUI:
Label widget: Used to display text instructions and information.

Entry Widget: Used for user input, such as first number and second number.

Button Widget: Used to trigger the calculation of all mathematic operation as described initially.

Code:

```
import tkinter as tk
from tkinter import messagebox
# function for maximum number
def find_biggest():
    # Try and except statement is used
    # for handling the errors like zero
    # division error etc.
    # Put that code inside the try block
    # which may generate the error
    # if error occurs during code execution
    # then it will be caught
    # and handle by the except block
    # if the user try to enter invalid number
    # error message will show
    try:
        num1 = float(entry1.get()) # Get first number
        num2 = float(entry2.get()) # Get second number
        biggest = max(num1, num2) # find maximum number between
2 numbers
        messagebox.showinfo("Result", f"The biggest number
between {num1} and {num2} is: {biggest}")
    except ValueError:
```

```
        messagebox.showerror("Error", "Please enter valid numbers.")
# function for minimum number
def find_smallest():
    try:
        num1 = float(entry1.get())
        num2 = float(entry2.get())
        smallest = min(num1, num2)
        messagebox.showinfo("Result", f"The smallest number
between {num1} and {num2} is: {smallest}")
    except ValueError:
        messagebox.showerror("Error", "Please enter valid numbers.")
# function for addition
def calculate_sum():
    try:
        num1 = float(entry1.get())
        num2 = float(entry2.get())
        sum_nums = num1 + num2
        messagebox.showinfo("Result", f"The sum of {num1} and
{num2} is: {sum_nums}")
    except ValueError:
        messagebox.showerror("Error", "Please enter valid numbers.")
# function for subtraction
def calculate_diff():
    try:
        num1 = float(entry1.get())
        num2 = float(entry2.get())
        diff_nums = num1 - num2
        messagebox.showinfo("Result", f"The difference   between
{num1} and {num2} is: {diff_nums}")
    except ValueError:
        messagebox.showerror("Error", "Please enter valid numbers.")
# function for multiplication
def calculate_product():
    try:
```

```
        num1 = float(entry1.get())
        num2 = float(entry2.get())
        product = num1 * num2
        messagebox.showinfo("Result", f"The product of {num1} and
{num2} is: {product}")
    except ValueError:
        messagebox.showerror("Error", "Please enter valid numbers.")
# function for division
def calculate_div():
    try:
        num1 = float(entry1.get())
        num2 = float(entry2.get())
        # Check second number is equal to zero
        # if it is, then value error raised
        # it it is not, then calculate the quotient
        if num2 == 0:
            raise ValueError("Division by zero is not allowed.")
        div_nums = num1 / num2
        messagebox.showinfo("Result", f"The division of {num1} and
{num2} is: {div_nums}")
    except ValueError as e:
        messagebox.showerror("Error", str(e))
# function for modulo operation
def calculate_modulo():
    try:
        num1 = float(entry1.get())
        num2 = float(entry2.get())
        mod_nums = num1 % num2
        messagebox.showinfo("Result", f"The remainder of {num1}
and {num2} is: {mod_nums}")
    except ValueError:
        messagebox.showerror("Error", "Please enter valid numbers.")
# Create the main window
root = tk.Tk()
```

```
root.title("Number Operations")
# Create Label for Project title
# Apply font style to title label
font_bold = ("Helvetica",18,"bold")
label_title = tk.Label(root, text="Simple Math Operations Calculator")
label_title.config(fg="white",bg="Dark Green",font=font_bold)
label_title.grid(row=0, columnspan=3, padx=5, pady=5)
# Create and place widgets
label1 = tk.Label(root, text="Enter first number:")
label1.config(font=("Arial",10,"bold"))
label1.grid(row=1, column=1, padx=5, pady=5)
entry1 = tk.Entry(root)
entry1.grid(row=1, column=2, padx=5, pady=5)
label2 = tk.Label(root, text="Enter second number:")
label2.config(font=("Arial",10,"bold"))
label2.grid(row=2, column=1, padx=5, pady=5)
entry2 = tk.Entry(root)
entry2.grid(row=2, column=2, padx=5, pady=5)
biggest_button = tk.Button(root, text="Find Biggest",
command=find_biggest)
biggest_button.config(font=("Helvetica",10,"bold"))
biggest_button.grid(row=3, column=1, padx=5, pady=5)
smallest_button = tk.Button(root, text="Find smallest",
command=find_smallest)
smallest_button.config(font=("Helvetica",10,"bold"))
smallest_button.grid(row=3, column=2, padx=5, pady=5)
# Create button frame
button_frame = tk.Frame(root)
button_frame.grid(row=4, column=1, columnspan=2, padx=5,
pady=5)
# Create buttons inside the button frame
# Create sum button to find addition of two numbers
sum_button = tk.Button(button_frame, text="Sum",
command=calculate_sum)
```

```
sum_button.grid(row=0, column=0, padx=5, pady=5)
sum_button.config(font=("Helvetica", 10, "bold"))
# Create diff button to find difference between  two numbers
diff_button = tk.Button(button_frame, text="Difference",
command=calculate_diff)
diff_button.grid(row=0, column=1, padx=5, pady=5)
diff_button.config(font=("Helvetica", 10, "bold"))
# Create product button to find product of two numbers
product_button = tk.Button(button_frame, text="Product",
command=calculate_product)
product_button.grid(row=0, column=2, padx=5, pady=5)
# the button's text will be displayed in a bold Helvetica font with
a size of 10
product_button.config(font=("Helvetica", 10, "bold"))
# Create div button to find quotient
div_button = tk.Button(button_frame, text="Division",
command=calculate_div)
div_button.grid(row=0, column=3, padx=5, pady=5)
div_button.config(font=("Helvetica", 10, "bold"))
# Create mod button to find remainder
mod_button = tk.Button(button_frame, text="Modulo",
command=calculate_modulo)
mod_button.grid(row=0, column=4, padx=5, pady=5)
mod_button.config(font=("Helvetica", 10, "bold"))
# Start the Tkinter event loop
root.mainloop()
```

Design of GUI:

After executing the provided code, a graphical user interface (GUI) window will appear on your screen. Let’s see what you will see in GUI:

Input Textboxes: At the top of the calculator, there are two textboxes where users can enter their numerical values. One box is labeled "Enter First Number" for inputting the first number, and the

other is labeled " Enter Second Number" for inputting the second number.

Operation Buttons: Below the input textboxes, there are seven buttons representing different mathematical operations:

Sum : Calculates the sum of the two numbers.

Difference: Calculates the difference between the two numbers.

Product: Calculates the product of the two numbers.

Division: Calculates the quotient of the division operation.

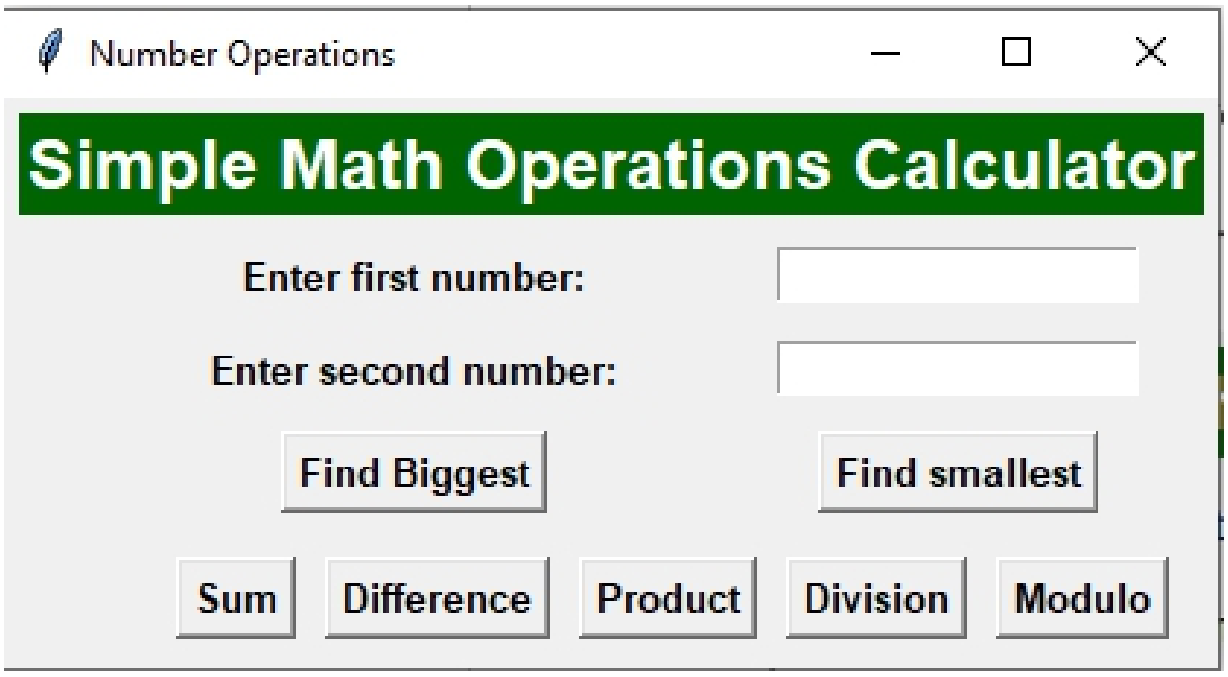
Modulo: Calculates the remainder of the division operation.

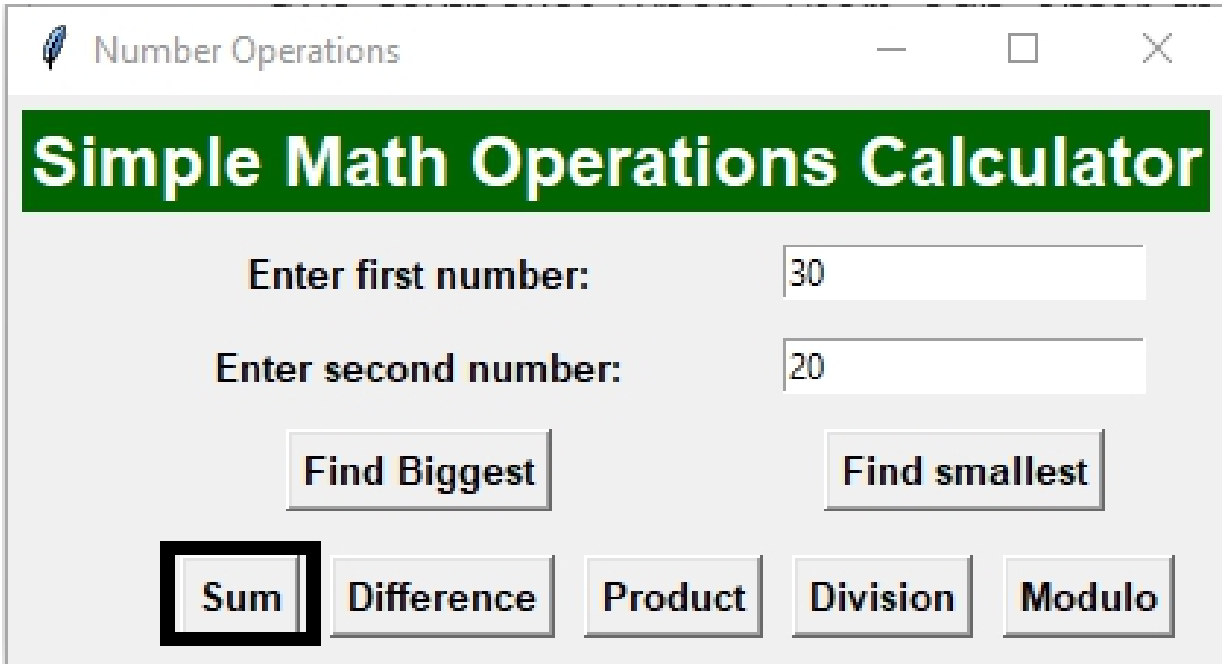
Find Biggest: Finds the biggest number among the two.

Find Smallest: Finds the smallest number among the two.

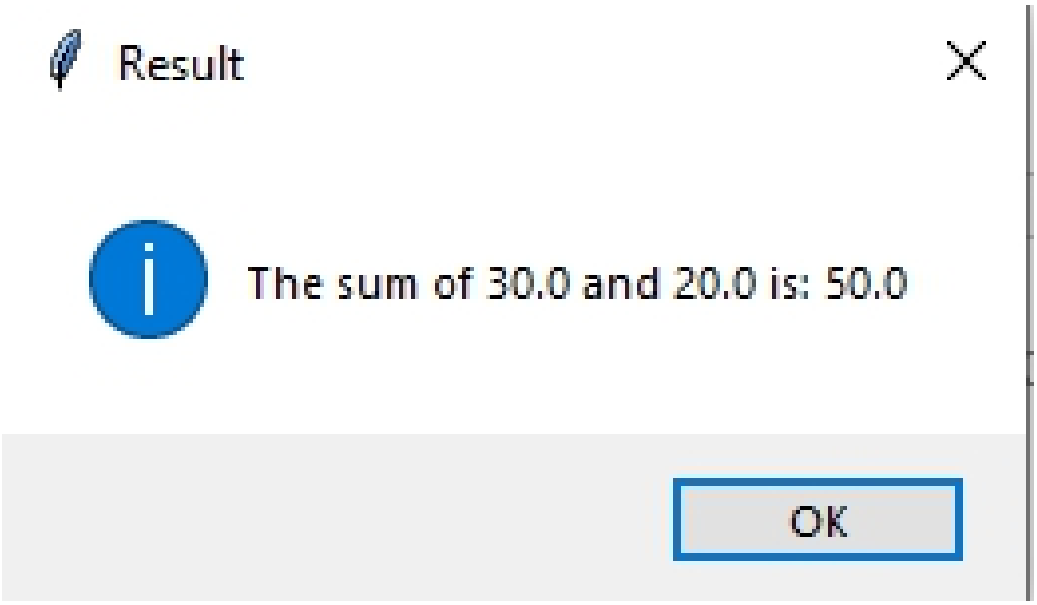
Result Display: Upon clicking any of the operation buttons, the calculator performs the corresponding mathematical operation using the input numbers. The calculated result is then displayed in a messagebox.

Output:
After running the provided code, you'll see a graphical user interface (GUI) window displayed on your screen.





After clicking the ‘Sum’ button, following messagebox is displayed on your screen.



Note:

My book, titled 'Python Tkinter: 35 Mini Projects,' offers 35 mini projects with accurately compiled code. It also explains how to create Tkinter widgets and how to use them to develop amazing Python GUI projects. Feel free to check it out to understand more.

Pyramid Pattern Generator

Project #2

Design a Simple GUI using Tkinter in Python to generate different pyramid pattern.

Introduction:

Welcome to the Pyramid Pattern Generator project! In this project, we'll design a simple Graphical User Interface (GUI) using Tkinter to display different pyramid patterns. As you know, Tkinter is a popular Python library for building desktop applications.

Our goal is to design an application where users can generate pyramid patterns using letters, numbers, or symbols. With Tkinter, we will create a simple and easy-to-use interface, allowing users to select from alphabet, number, and symbol patterns option. They'll be able to enter the number of rows they want for their pyramid, then click on the “Generate pattern” button to see the pattern generated.

Let's get started and create this fun tool together!

Requirments:

Tkinter Widgets: We'll use four Tkinter widgets to design the GUI:
Label widget: Used to display text instructions and information.

Entry Widget: Used for user input, such as row number.

Button Widget: Used to intiate the pattern generation process as described initially.

Option button Widget : Used to select the pyramid pattern type

Text Area Widget : Used to see generated pattern

Let’s try to design this GUI

Code:

```
import tkinter as tk

def generate_pattern():
    # Get user input from option widget
    pattern_type = var.get()
    # Get user input from entry widget
    rows = int(entry_rows.get())
    # Intialize empty string to store generated pattern
    pattern_text = ""
    # Check the user input and generate the corrsponding
pyramid pattern
    # if user input is 1 then generate alphabet pattern
    if pattern_type == 1:  # Alphabet
        for i in range(rows):
     # Add spaces to align pattern based on current row
            pattern_text += " " * (rows - i - 1)
            for j in range(i + 1):
                # Add corresponding alphabet character
                # to pattern_text using ASCII values
                pattern_text += chr(65 + j) + " "
            # Add new line character to move to next row
            pattern_text += "\n"
    # if user input is 2 then generate number pattern
    elif pattern_type == 2:  # Number
        for i in range(rows):
     # Add spaces to align pattern based on current row
            pattern_text += " " * (rows - i - 1)
            for j in range(i + 1):
                # Add corresponding number to current row
                pattern_text += str(j + 1) + " "
                # Add new line character to move to the next row
            pattern_text += "\n"
    # if user input is 3 then generate symbol pattern
    elif pattern_type == 3:  # Symbol
```

```
        for i in range(rows):
    # Add spaces to align pattern based on current row
            pattern_text += " " * (rows - i - 1)
            for j in range(i + 1):
                # Add * symbol to the pattern_text
                pattern_text += "* "
        # Add new line character to move to the next row
            pattern_text += "\n"
    # Clear the previous result if any
    output_text.delete(1.0, tk.END)
    # Insert new result in text area
    output_text.insert(tk.END, pattern_text)

# Create the main window
root = tk.Tk()
root.title("Pattern Generator")
# Create Label for title
label_title = tk.Label(root, text="Pyramid Pattern Generator")
label_title.pack()
label_title.config(fg="white", bg="blue", font=("Helvetica", 18,"bold"))
# Create a frame for options
options_frame = tk.Frame(root)
options_frame.pack()

# Option buttons
var = tk.IntVar()
# Create option button for alphabet
alphabet_btn = tk.Radiobutton(options_frame, text="Alphabet",
variable=var, value=1)
alphabet_btn.pack(side=tk.LEFT, padx=5)
# Create option button for number
number_btn = tk.Radiobutton(options_frame, text="Number",
variable=var, value=2)
number_btn.pack(side=tk.LEFT, padx=5)
```

```
# Create option button for Symbol "*"
symbol_btn = tk.Radiobutton(options_frame, text="Symbol",
variable=var, value=3)
symbol_btn.pack(side=tk.LEFT, padx=5)

# Create label for Rows entry
rows_label = tk.Label(root, text="Enter number of rows:")
rows_label.pack()
rows_label.config(font=("Arial", 10, "bold"))
# Create Entry widget for user to input number of rows
entry_rows = tk.Entry(root)
entry_rows.pack(pady=5)

# Generate button
generate_btn = tk.Button(root, text="Generate Pattern",
command=generate_pattern)
generate_btn.pack(pady=5)
generate_btn.config(fg="blue", bg="white", font=("Arial",12,"bold"))

# Create Text Area widget for result
output_text = tk.Text(root, height=10, width=30)
output_text.pack(fill=tk.BOTH, expand=True)
root.mainloop()
```

Design of GUI :

After executing the provided code, a graphical user interface (GUI) window will appear on your screen. Let’s see what you will see in GUI:

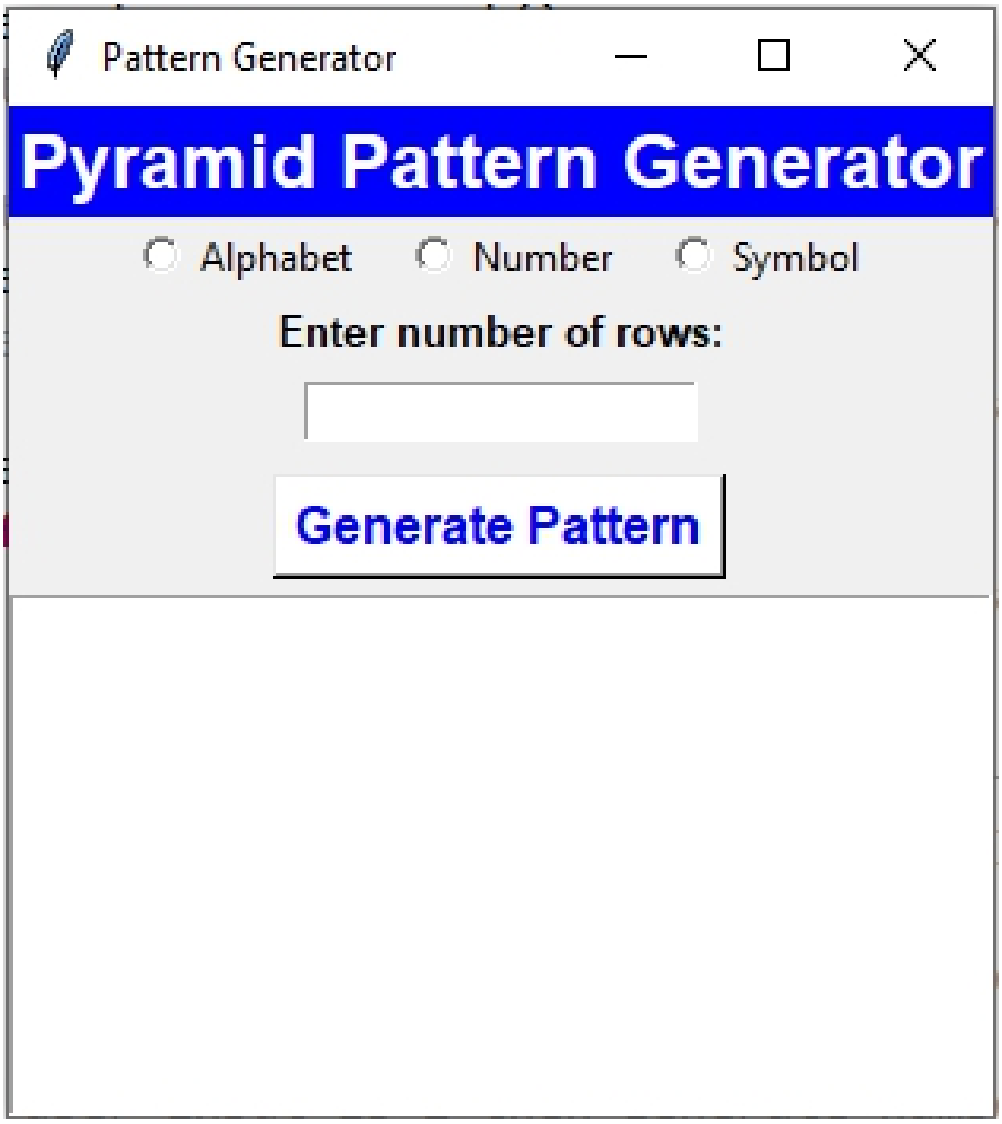
Option Buttons for Pattern Type: At the top of window, There are three option buttons labeled "Alphabet," "Number," and "Symbol." These buttons allow users to select the type of pattern they want to generate: alphabet patterns, number patterns, or symbol patterns.

Label and Entry Box for Number of Rows: Beneath the option buttons, there is an entry box labeled “Enter number of rows “ where users can input the number of rows they want for their pyramid pattern.

"Generate Pattern" Button: Below the entry box, there is a button labeled "Generate Pattern." Users can click on this button to initiate the pattern generation process. Once clicked, result of generated pattern is shown in text Area.

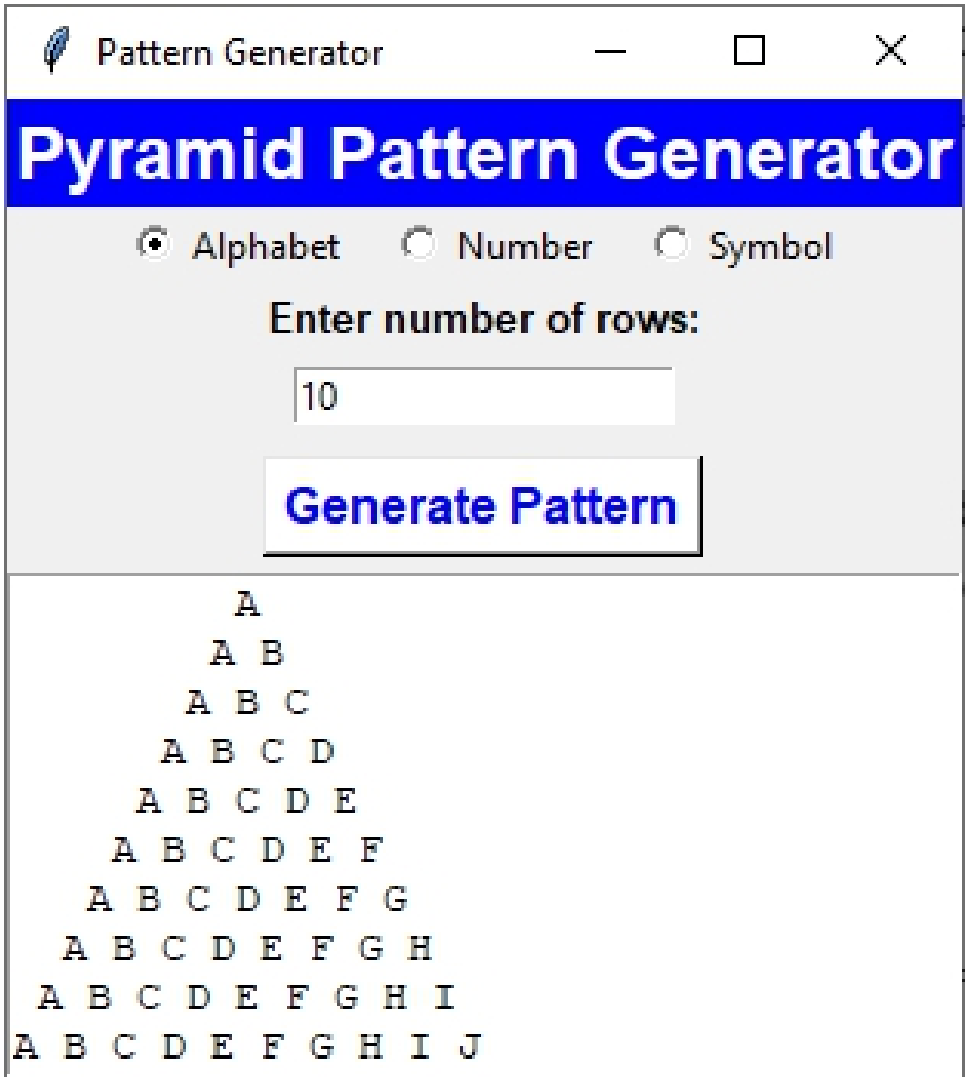
Output:

After running the provided code, you'll see a graphical user interface (GUI) window displayed on your screen.

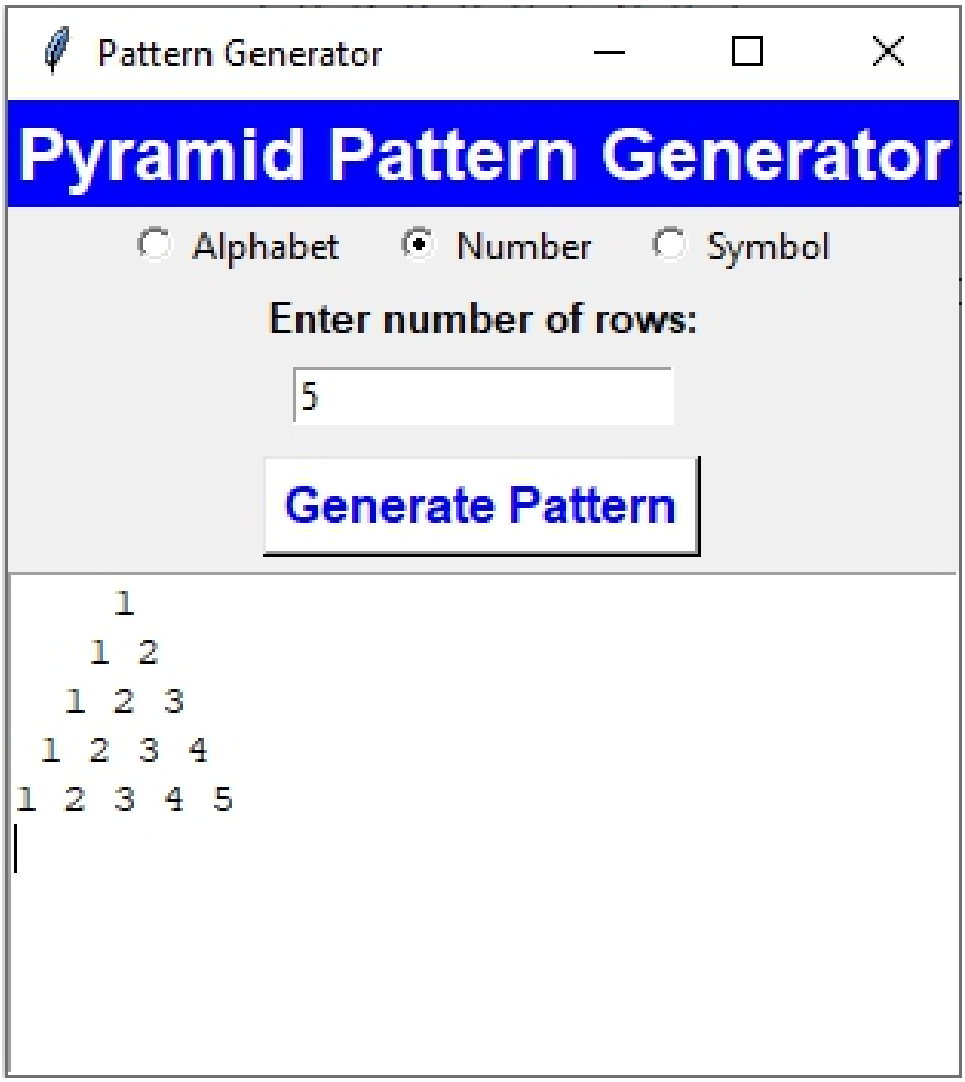


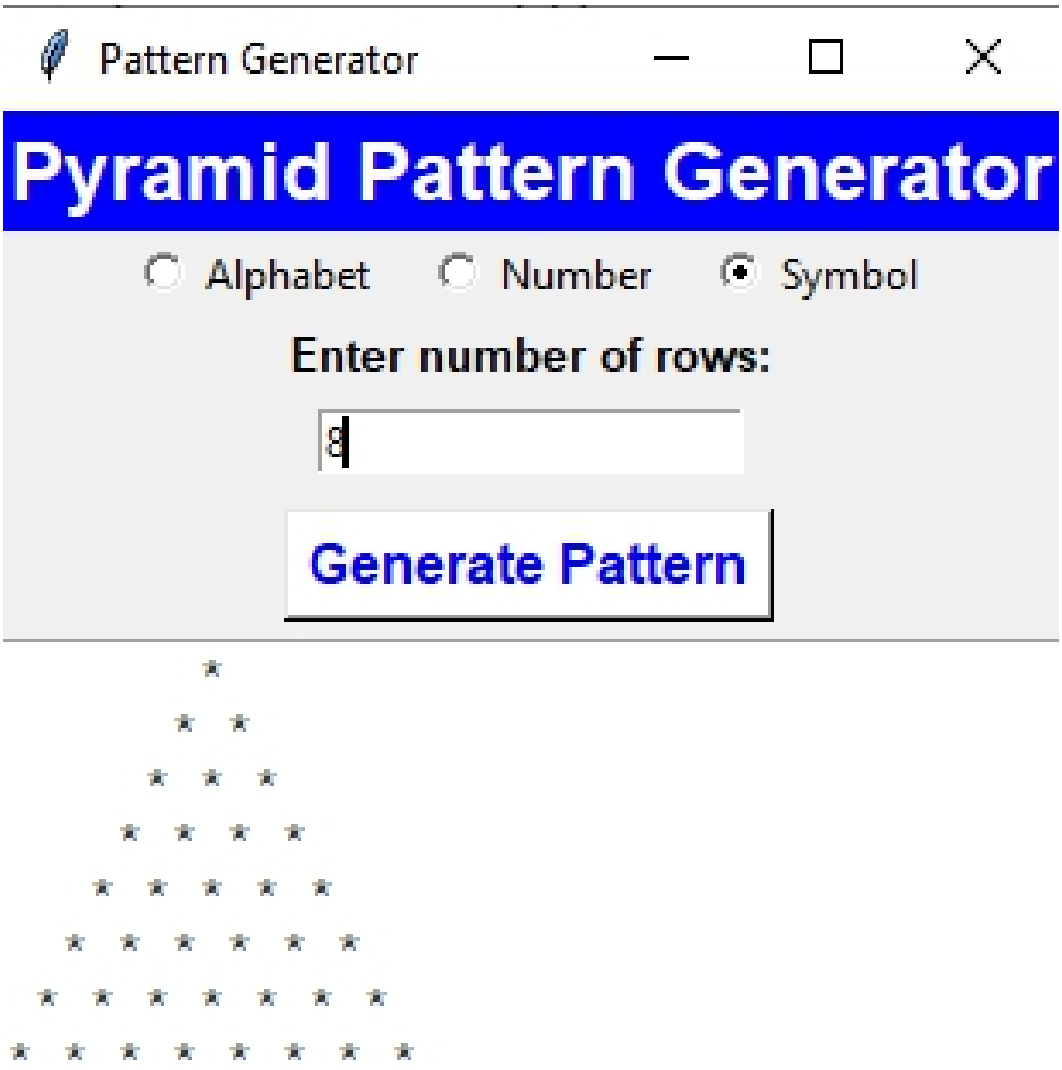
This is the main GUI window, where the user can select a pyramid pattern by clicking on option buttons and enter the number of rows to

generate the selected pyramid pattern with the click of the Generate Pattern button.



After selecting the Alphabet option and entering 10 rows, clicking the Generate Pattern button generated a 10-row Alphabet pyramid, as shown above.


Pattern Generator
Pyramid Pattern Generator
Alphabet
Number
Symbol
Enter number of rows:
5
Generate Pattern
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5



Here, the user selects the Symbol option, enters 8 rows, and clicks on the Generate Pattern button. This action generates an 8-row Symbol pyramid pattern, as depicted in the output.

Area and Perimeter Calculator for Geometric Shapes

Project #3

Creating a Tkinter GUI for Area and Perimeter Calculator for Geometric shapes

Introduction:

Welcome to the Geometric Shape Area and Perimeter Calculator project! In this project, we'll explore how to create a simple geometric shape area and perimeter calculator using Python Tkinter. Our aim is to build an application that allows users to calculate the area and perimeter of shapes like squares, rectangles, and triangles.

To accomplish this, we'll use Tkinter, the most popular library for creating GUIs in Python. We'll design a user-friendly interface where users can select the shape they want to work with from a dropdown box.

Depending on their selection, the necessary input fields will automatically appear on the main window. For example, if a user chooses a square, they'll see a Entry box where they can enter the length of one of its sides. After entering the required information, users can click the "Calculate" button to instantly calculate the area and the perimeter of the selected shape. Let's dive in and create our Tkinter project!

Requirements:

Tkinter Widgets: We'll use Four Tkinter widgets to design the GUI:
Dropdown box or Combo box : Used for shape selection

Label widget: Used to display text instructions and information.

Entry Widget: Used for user input, such as side length, length, width, base, and height.

Button Widget: Used to trigger the calculation of area and perimeter.

Before writing the code, it's essential to understand the concepts of Area and Perimeter of Geometric shapes and how to calculate them.

The **perimeter** is the distance around the outside of a shape. **For example** : Imagine you are putting a fence around a rectangular playground. To find out how much fencing material you need, you would calculate the perimeter of the playground.

The **area** is the amount of space inside a shape. It’s useful for figuring out how much material you need to cover a surface, **For Example** If you want to tile your kitchen floor, you need to know the area to determine how many tiles to purchase. If your kitchen is rectangular, you would multiply the length by the width to get the area.

Area and Perimeter Formulas for Geometric shapes

Rectangle:

Area of Rectangle = length x width

Perimeter of Rectangle =2 × (length+width)

Example :

If the length is 10 meters and the width is 5 meters,

Area = 10m x 5m = $50m^2$

Perimeter = 2 x (10m + 5m) = 2 x 15m = 30m

Square:

Area of Square = side x side

Perimeter of Square =4 × side

Example : If each side is 4 meters, then:

Area = 4m x 4m = 16m2

Perimeter = 4m x 4m = 16m2

Triangle:

Area of Triangle = ½ (base x height)

Perimeter of Triangle = side1 + side2 + side3

Example :

If the base is 6 meters and the height is 3 meters, then:

Area = ½ x (6m x 3m) = ½ (18m2) = 9m2

Perimeter = 6m + 6m + 6m = 18m

We will use all this formulas in our code to calculate area and perimeter of square,rectangle and triangle.

Lets implement the code

Code :

```
import tkinter as tk

# Calculate function

def calculate():

    shape = shape_var.get()

    # Get the relevant entry widgets based on the selected shape

    relevant_entry_widgets = entry_widgets[shape]

    # Extract values from the entry widgets

    # Initialize an empty list to store the extracted values
```

```
    values = []
    # Iterate over the relevant entry widgets,
    #starting from index 1 and skipping every other widget
    for i in range(1, len(relevant_entry_widgets), 2):# we skip the label and iterate over entry widget only
        # Access the entry widget at the current index
        entry_widget = relevant_entry_widgets[i]
        # Retrieve the value from the entry widget and convert it to a float
        value = float(entry_widget.get())
        # Append the extracted value to the list of values
        values.append(value)
   # Check formulas for all shapes
    if shape == "Square": # if selected shape is square
        side_length = values[0]
        area = side_length ** 2
        perimeter = 4 * side_length
    elif shape == "Rectangle":# if selected shape is rectangle
        length, width = values
        area = length * width
        perimeter = 2 * (length + width)
    elif shape == "Triangle":# if selected shape is triangle
```

```
        base, height = values
        area = 0.5 * base * height
        # Assuming it's an equilateral triangle for simplicity
        perimeter = 3 * base
    else:
        area = perimeter = 0  # Default values for unknown shape
    result_label.config(text=f"Area: {area:.2f}\nPerimeter: {perimeter:.2f}")
# Create show variable function
def show_variables(*args):
    # Retrieve the currently selected shape from the shape variable
    shape = shape_var.get()
    # Hide all entry widgets and labels initially
    for widget_list in entry_widgets.values(): # All widgets are stored in entry_widgets dictionary
        for widget in widget_list:
            widget.grid_remove() # Remove all entry and label widget from grid layout
    # Display entry widgets and labels based on the selected shape
    for widget in entry_widgets[shape]:
        widget.grid() # Display selected shape on grid layout
```

 # Create the main window

root = tk.Tk()

root.title("Shape Calculator")

root.config(bg="white")

 # Create Label for Project title

 # Apply font style to title label

font_bold = ("Helvetica",18,"bold")

label_title = tk.Label(root, text="Area and Perimeter Calculator for Shapes")

label_title.config(fg="white",bg="Navy Blue",font=font_bold)

label_title.grid(row=0, columnspan=2, padx=5, pady=5)

 # Create and place widgets

 # Create label for shape

label_shape = tk.Label(root, text="Select a Shape:")

label_shape.config(font=("helvetica",15), bg="white", fg="Navy blue")

label_shape.grid(row=1, column=0, pady=5)

 # Create Option menu for shape

shapes = ["Square", "Rectangle", "Triangle"]

shape_var = tk.StringVar(value=shapes[0])

shape_menu = tk.OptionMenu(root, shape_var, *shapes, command=show_variables)

```
shape_menu.config(width=25, font=("TkDefaultFont", 10,
"bold"),fg="Navy Blue",bg="white")

shape_menu.grid(row=1, column=1, pady=5)

# Create entry widgets and labels for each shape dynamically

# store all widgets and labels in entry_widget dictionary

entry_widgets = {

    "Square": [tk.Label(root, text="Side Length:"), tk.Entry(root)],

    "Rectangle": [tk.Label(root, text="Length:"), tk.Entry(root),
tk.Label(root, text="Width:"), tk.Entry(root)],

    "Triangle": [tk.Label(root, text="Base:"), tk.Entry(root),

                tk.Label(root, text="Height:"), tk.Entry(root)]

}

# Place entry widgets and labels on the form and initially hide
them

# Initialize the row count for grid layout to 2

row_count = 2

# Loop through each shape in the list of shapes

for shape in shapes:

    # Iterate over each widget in the list of widgets
corresponding to the current shape

    for widget in entry_widgets[shape]:

        # Place the widget on the grid layout with specified row,
columnspan, and padding
```

```
        widget.grid(row=row_count, columnspan=2, pady=5)

        # Increment the row count to move to the next row in the grid layout

        row_count += 1

        # Initially hide the widgets by removing them from the grid layout

        widget.grid_remove()

# Create calculate button

calculate_button = tk.Button(root, text="Calculate", command=calculate,width=10, height=2)

calculate_button.config(font=("Arial", 12, "bold"),bg="Navy Blue",fg="white")

calculate_button.grid(row=row_count, column=0, columnspan=2, pady=10)

# Create result label

result_label = tk.Label(root, text="")

result_label.config(font=("Arial", 12, "bold"),bg="white",fg="Navy Blue")

result_label.grid(row=row_count + 1, column=0, columnspan=2, pady=10)

# Start the Tkinter event loop

root.mainloop()
```

Design of GUI:

The design of the Geometric Shape Area and Perimeter Calculator project focuses on creating a user-friendly interface that allows users

to easily interact with the application and perform calculations for different geometric shapes.

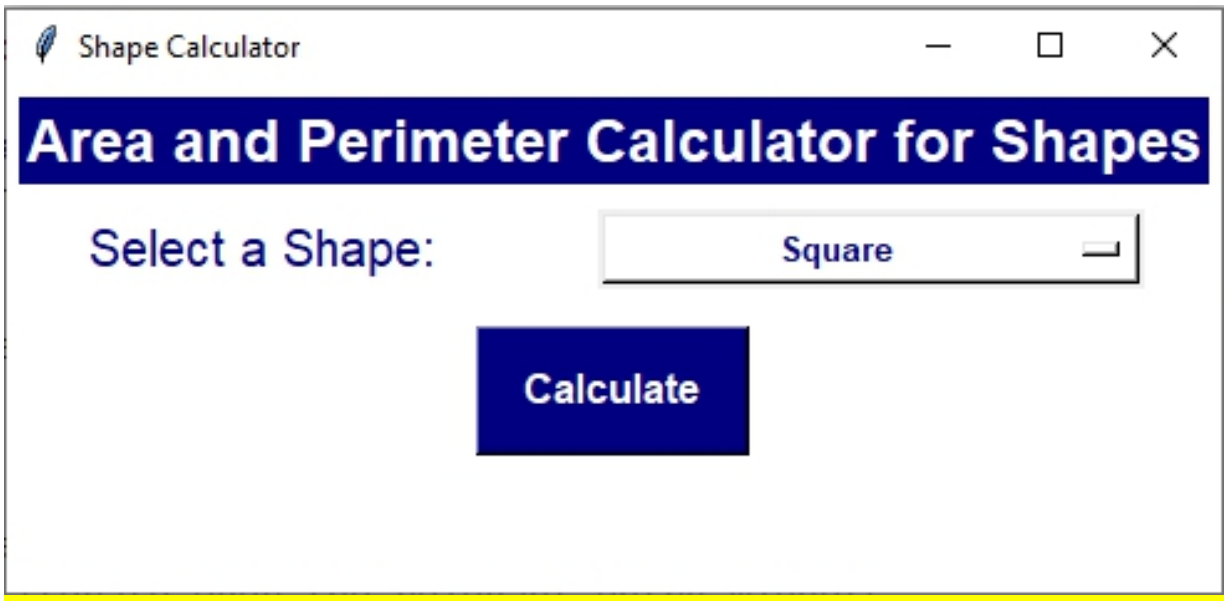
Shape Selection: At the top of the UI, users can find a dropdown box labeled "Select Shape." This dropdown allows users to choose the geometric shape for which they want to calculate the area and perimeter.

Input Fields: Below the shape selection dropdown, the UI dynamically displays input fields based on the selected shape. For example, if the user selects a triangle, two input fields labeled "Base" and “Height” appears. Similarly, if the user selects a rectangle, two input fields labeled "Length" and "Width" appear. This dynamic layout ensures that users only see the relevant input fields for the selected shape, simplifying the input process.

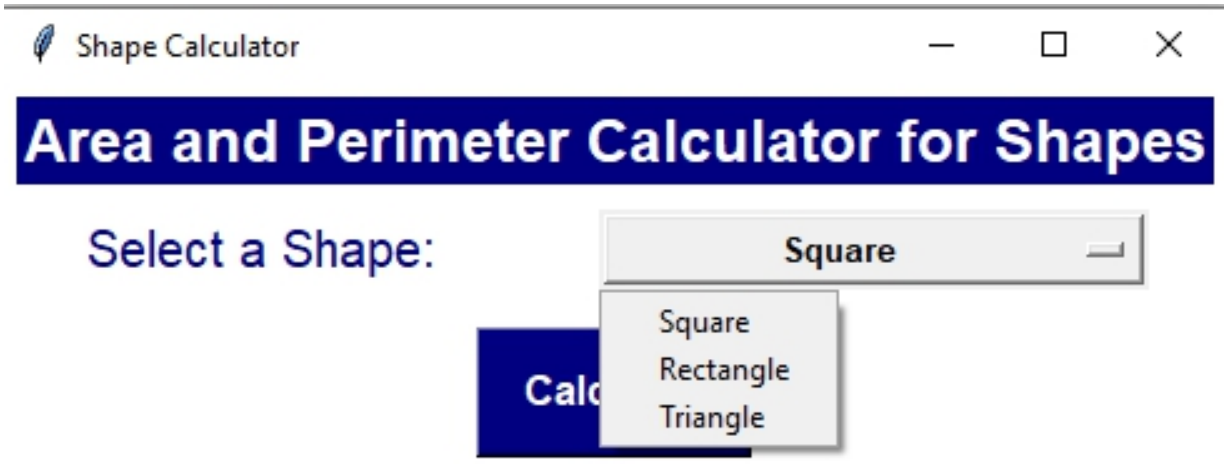
Calculation Button: A "Calculate" button is provided below the input fields. Users can click this button to trigger the calculation of the area and perimeter based on the input values provided.

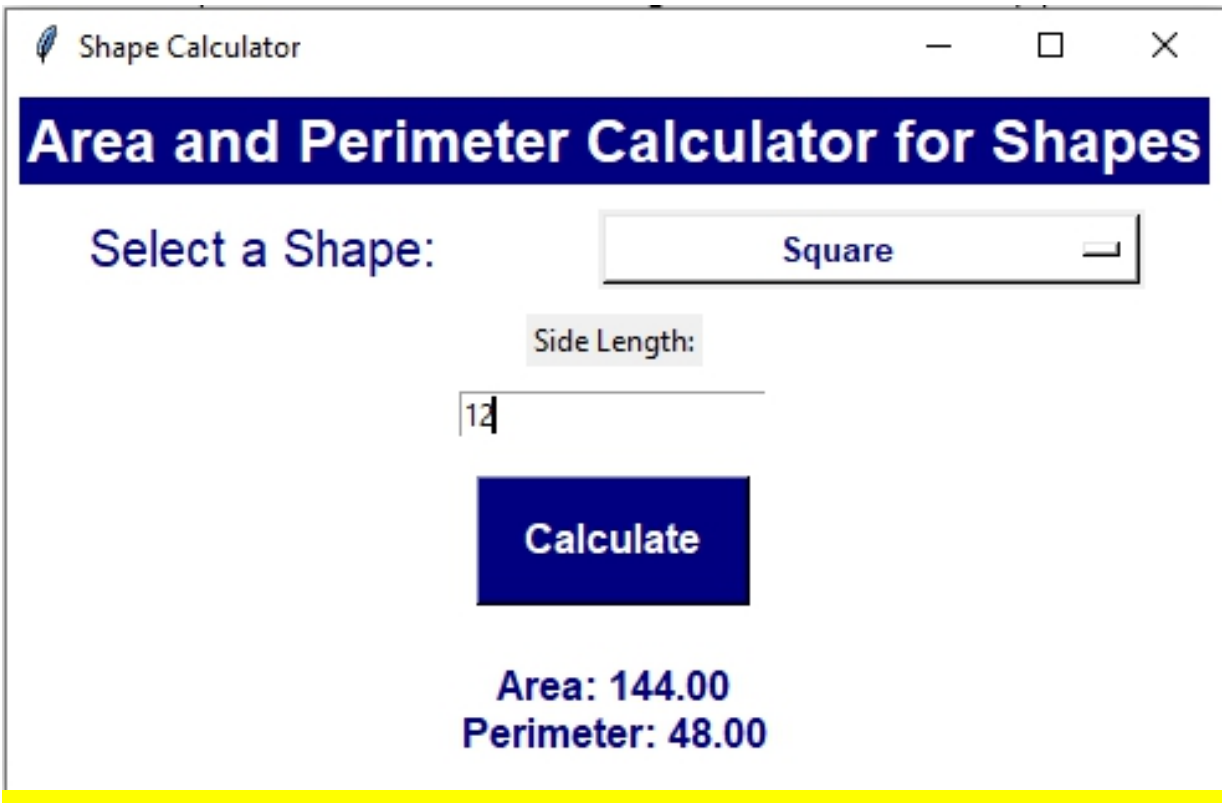
Results Display: The calculated area and perimeter are displayed on the UI below the "Calculate" button.

Output:
After running the provided code, you'll see a graphical user interface (GUI) window displayed on your screen.

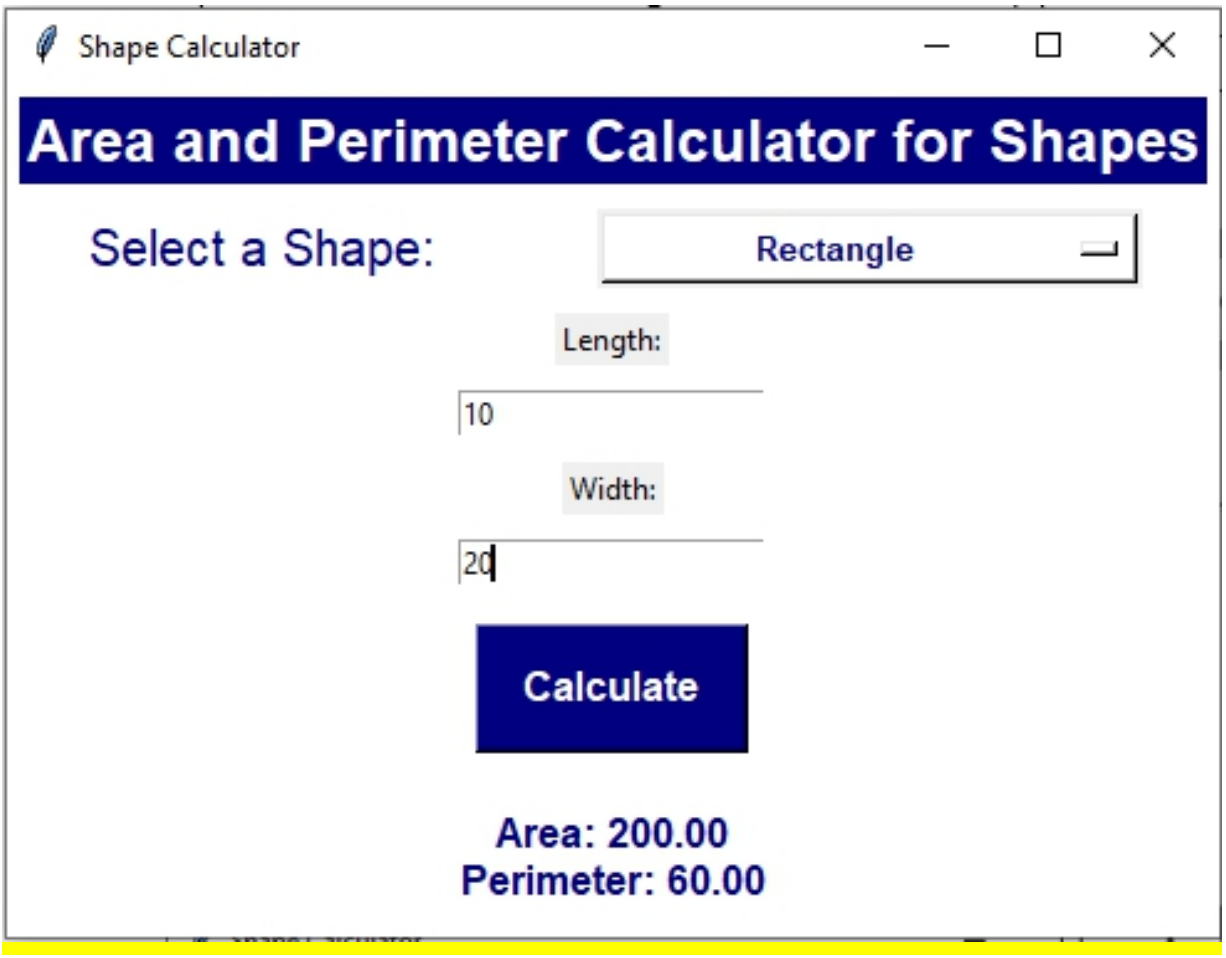


This is the main GUI window where the user can select shapes from a dropdown menu and calculate the area and perimeter of the selected shape by clicking the "Calculate" button.





When the user chooses the 'square' shape option from the dropdown menu, the side length label and its entry widget are automatically shown on the form, as illustrated above. Subsequently, if the user inputs a value of 12 for the side and clicks the calculate button, the calculated area and perimeter results for the square shape are displayed beneath the button, as indicated above.


Shape Calculator
Area and Perimeter Calculator for Shapes
Select a Shape:
Rectangle
Length:
10
Width:
20
Calculate
Area: 200.00
Perimeter: 60.00

Text Style Customizer

Project #4

Design a simple GUI form using Tkinter to customize fontstyle of text dynamically

Introduction:

In this project, we'll develop a user-friendly GUI application called Text Styler. Our objective is to create a application where users can easily customize the appearance of text. Text styling is essential for various purposes, and our aim is to simplify this task.

With Tkinter, we'll design an intuitive form that allows users to input text, select different text styles, adjust font sizes, and customize text colors and background colors. The goal is to empower users to personalize their text effortlessly.

Design of GUI :

To Design the GUI for this project, we will use following widget to make GUI more responsive and user friendly.

Text Input Field: At the top of the GUI, we'll include a text input field where users can enter the text they want to style.

Text Case Selection: Below the text input field, we'll place a dropdown menu (Combobox) that allows users to select from various text cases, including Uppercase, Lowercase, Capitalize Each Word, Sentence Case, and Toggle Case.

Font Size Adjustment: Adjacent to the text case selection, there will be a scrollbar widget that enables users to adjust the font size of the text dynamically. Users can slide the scrollbar to increase or decrease the font size according to their preferences.

Text Color and Background Color Buttons: Below the font size scrollbar, there will be two buttons: one for selecting the text color

and another for selecting the background color. When users click on these buttons, color dialog boxes will appear, allowing them to choose the desired colors. Once selected, the chosen colors will be applied to the text and background simultaneously.

Real-time Preview: As users make changes such as selecting a text case, adjusting font size, or choosing colors, the text displayed in the input field will update in real-time to reflect these modifications.

Let’s write the code now

Code:

```
import tkinter as tk
from tkinter import ttk
# imports the askcolor function from the colorchooser module
# The askcolor function allows users to select
# a color through a dialog box.
from tkinter.colorchooser import askcolor
# Toggle case function
def toggle_case(text):
    # Initializes an empty string variable
    # to store the toggled text.
    toggled_text = ""
    # Initiates a loop that iterates over
    # each character in the input text
    for char in text:
        # Checks if the current character is lowercase
        if char.islower():
        # If it is, appends its uppercase equivalent
        # to the toggled_text variable.
            toggled_text += char.upper()
        else:
            # If it is not, convert char into lowercase
            toggled_text += char.lower()
    # return result
```

```
    return toggled_text

# Capitalize each word function
def capitalize_each_word(text):
    capitalized_text = ""
    # text.split() is used to split the
    # input text into individual words based on spaces.
    for word in text.split():
        # During each cycle of the loop,
        # we capitalize the current word and
        # add it to our capitalized_text,
        # making sure to leave a space between each word.
        capitalized_text += word.capitalize() + " "
    return capitalized_text.strip()  # Remove trailing space

# Convert text function
def convert_text(text=None):
    selected_case = case_var.get()
    # Get text from the entry widget
    text = entry.get()
    # Initilize empty string to store converted text
    converted_text = ""
    # Check if the selected case is uppercase
    if selected_case == "Uppercase":
        # If it is, then convert entire text into uppercase
        # assign it to the variable 'converted_text'
        converted_text = text.upper()
        #  If the selected case is not "Uppercase",
        # checks if it's "Lowercase".
    elif selected_case == "Lowercase":
        # If the selected case is "Lowercase",
        # converts the entire text to lowercase and
        # assigns it to the variable converted_text.
        converted_text = text.lower()
    elif selected_case == "Sentence Case":
```

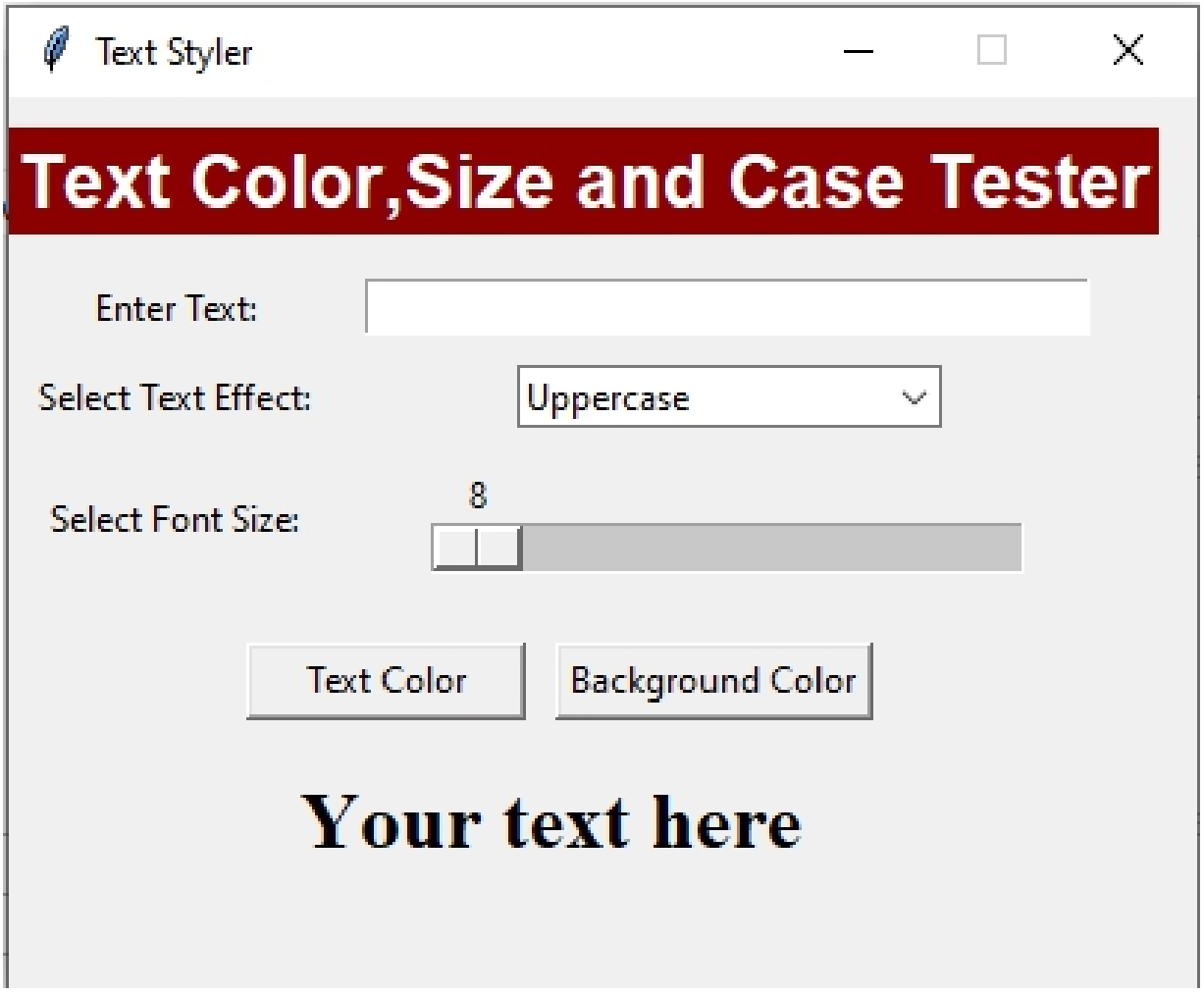
```
        converted_text = text.capitalize()
    elif selected_case == "Toggle Case":
        converted_text = toggle_case(text)
    elif selected_case == "Capitalize Each Word":
        converted_text = capitalize_each_word(text)
    label.config(text=converted_text)

def change_text_color():
    # Invokes the askcolor() function from the
tkinter.colorchooser
    # module to prompt the user to select a color.
    # The [1] index retrieves the selected color from the returned
tuple.
    color = askcolor()[1]
    # Change the text color
    label.config(fg=color)
# Function to change background color
def change_bg_color():
    color = askcolor()[1]
    # Change the background color
    label.config(bg=color)
# Function to change font size of text
def update_font_size(*args):
    # Retrieves the value from the scrollbar widget (scrollbar)
    # and converts it to an integer,
    # storing it in the variable font_size.
    font_size = int(scrollbar.get())
    label.config(font=("Arial", font_size))
# Create main window
root = tk.Tk()
root.title("Text Styler")
# Sets the initial size of the application window to
# 400 pixels in width and 300 pixels in height.
root.geometry("400x300")
```

```
# users cannot resize the application window
# either horizontally or vertically after it's been set to a specific size,
# effectively keeping its dimensions fixed.
root.resizable(False, False)

# Create title label
title_label = tk.Label(root, text="Text Color,Size and Case Tester")
title_label.grid(row=0, column=0, columnspan=3, pady=10)
title_label.config(fg="white", bg="Dark red", font=("Arial",19, "bold"))

# Create input entry
entry_label = tk.Label(root, text="Enter Text:")
entry_label.grid(row=1, column=0)
entry = tk.Entry(root, width=40)
entry.grid(row=1, column=1,pady=5)

case_label = tk.Label(root, text="Select Text Effect:")
case_label.grid(row=2, column=0)
# Create dropdown menu for text conversion
# Bind events to dynamically update the output
entry.bind("<KeyRelease>", convert_text)
case_var = tk.StringVar()
# Default selection
case_var.set("Uppercase")
# Create option list for combobox
case_options = ["Uppercase", "Lowercase", "Sentence Case",
"Capitalize Each Word", "Toggle Case"]
case_dropdown = ttk.Combobox(root, textvariable=case_var,
values=case_options)
case_dropdown.grid(row=2, column=1, padx=5,pady=5)
case_dropdown.bind("<<ComboboxSelected>>", convert_text)

# Create a horizontal scrollbar
scroll_label = tk.Label(root, text="Select Font Size:")
scroll_label.grid(row=3, column=0)
```

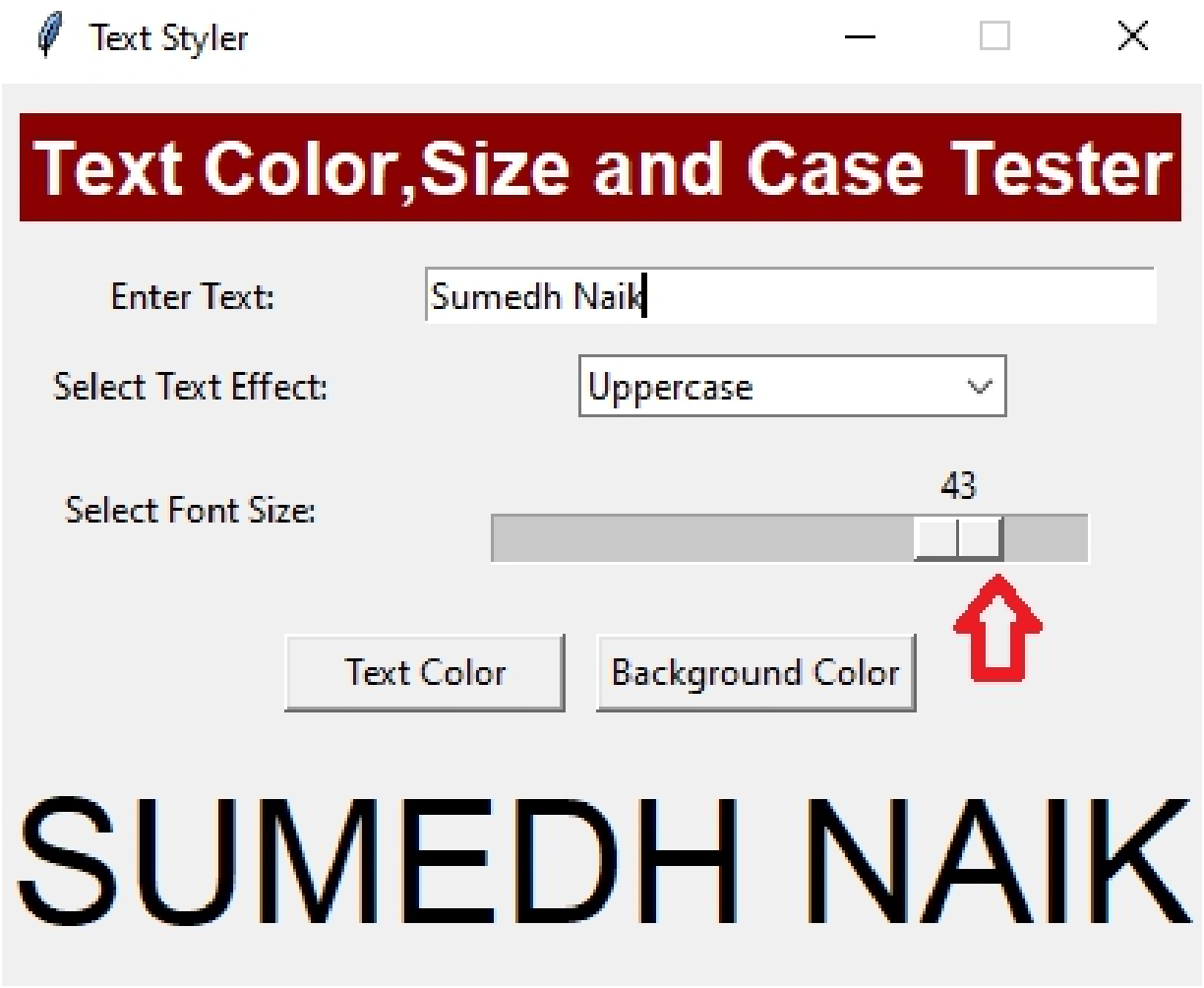
```
scrollbar = tk.Scale(root, orient=tk.HORIZONTAL,
command=update_font_size, from_=8, to=50, showvalue=1,
length=200)
scrollbar.grid(row=3, column=1, padx=5, pady=5)

# Create a frame for the buttons
button_frame = tk.Frame(root)
button_frame.grid(row=4, column=0, columnspan=2)

# Create buttons for changing text color and background color
text_color_button = tk.Button(button_frame, text="Text Color",
command=change_text_color, width=12)
text_color_button.grid(row=0, column=0, padx=5)
bg_color_button = tk.Button(button_frame, text="Background Color",
command=change_bg_color)
bg_color_button.grid(row=0, column=1, padx=5, pady=15)

# Create label for displaying converted text
label = tk.Label(root, text="Your text here ", font=("Times new
roman",20, "bold"))
label.grid(row=6, column=0, columnspan=2)

root.mainloop()
```

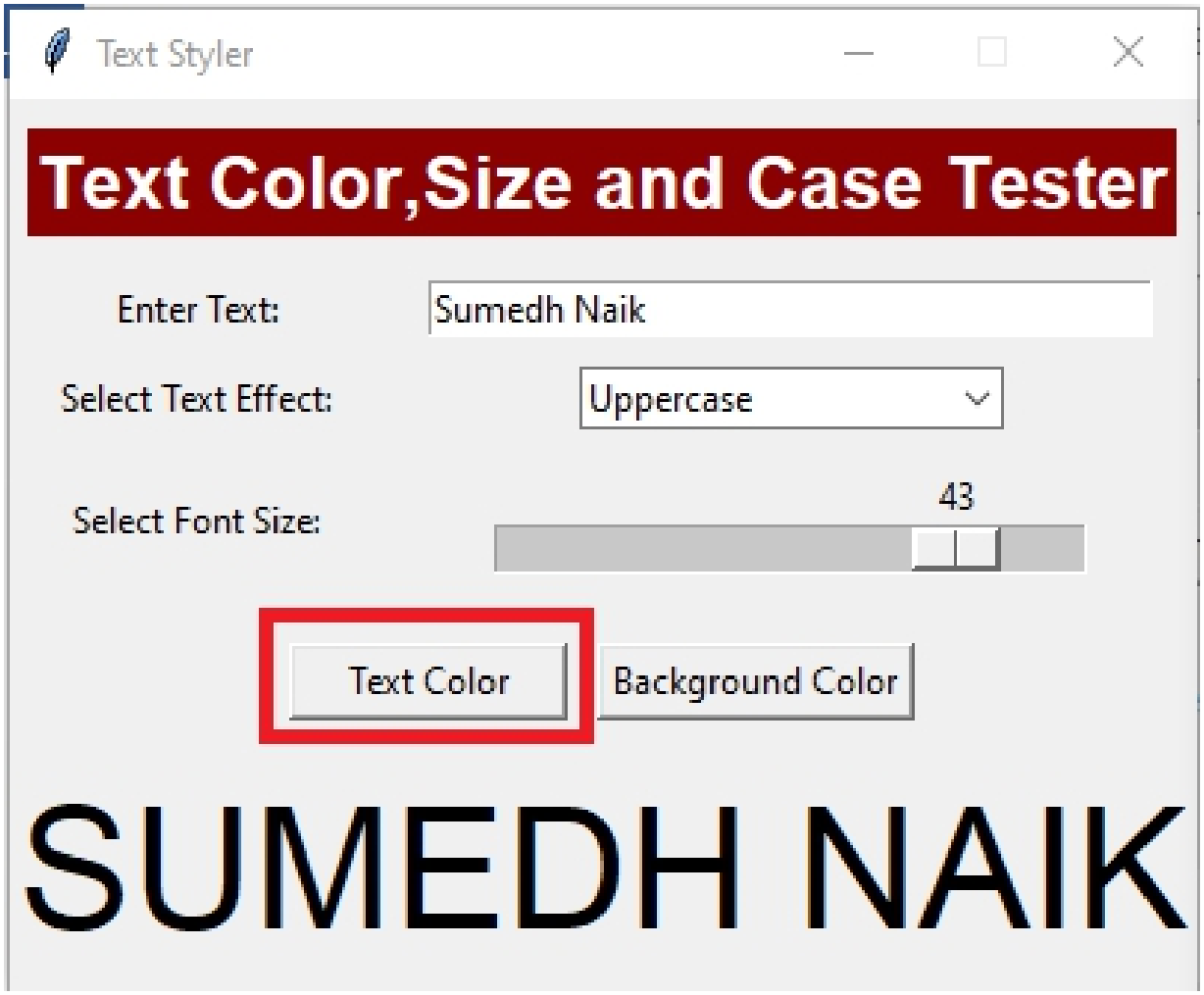
Output :

After running above code, following GUI will be displayed on your screen.

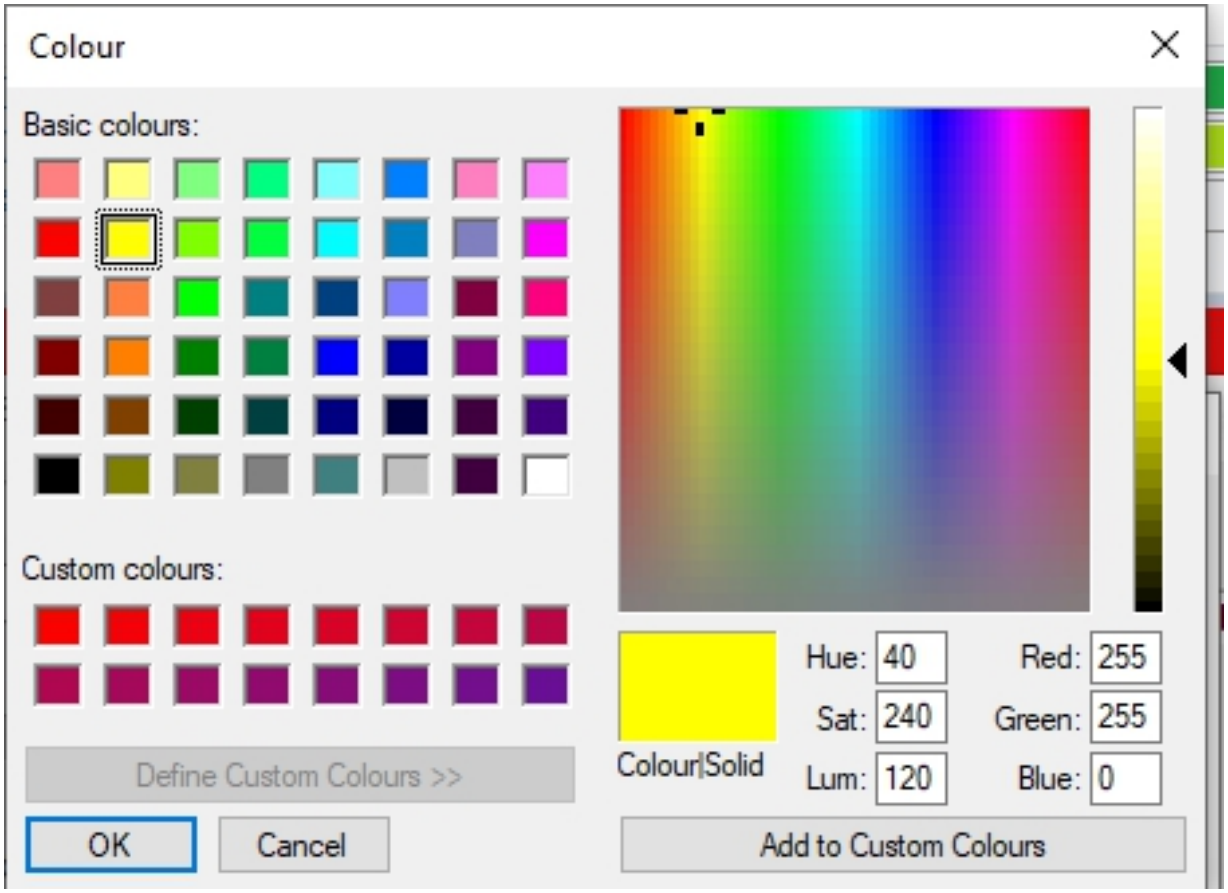


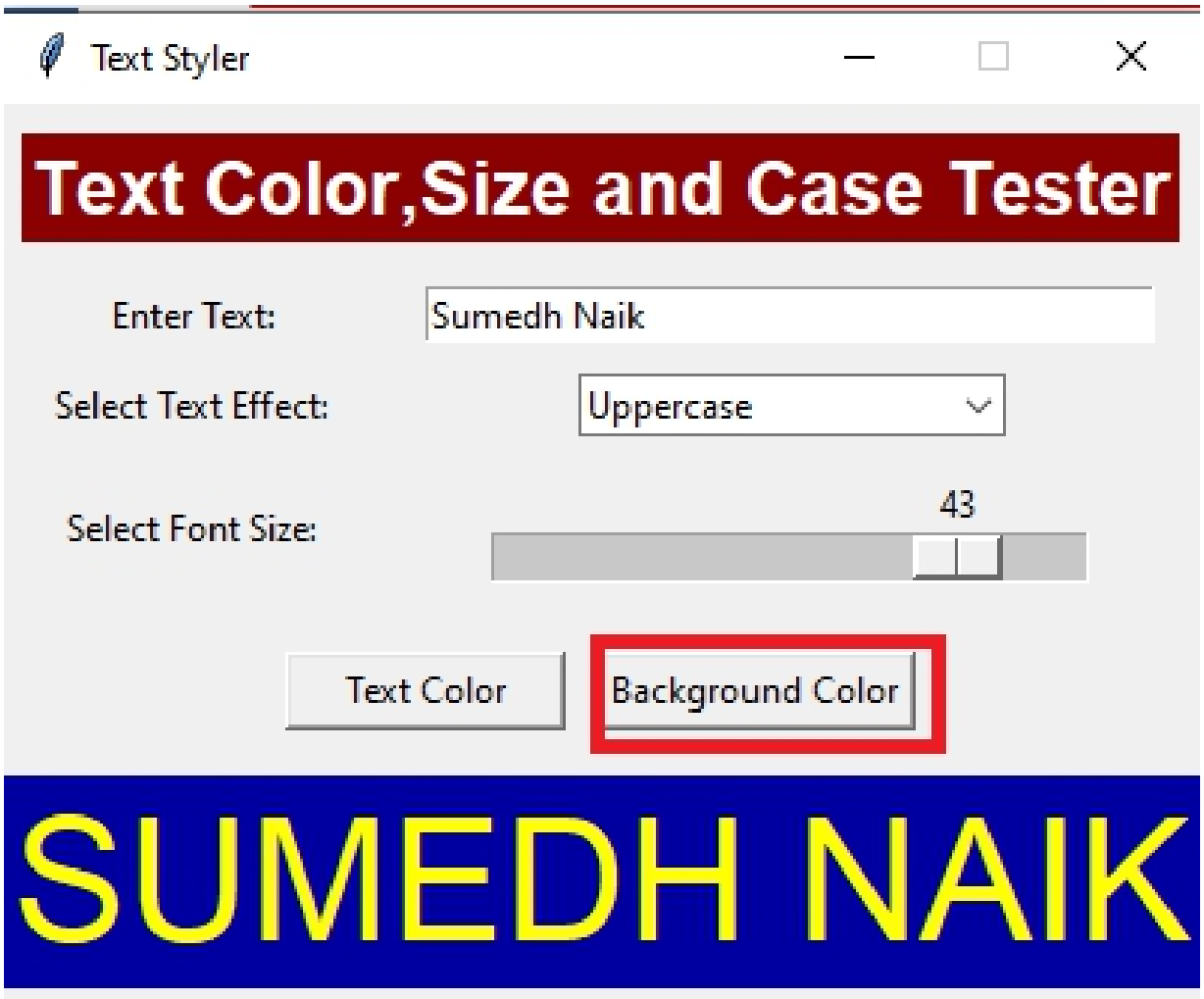
After inputting text into the entry box and selecting "uppercase" from the combobox, the label on the main window automatically updates to reflect the changes. Users can adjust the font size by sliding the scrollbar. As illustrated below, adjustments are made automatically.


Text Styler
Text Color,Size and Case Tester
Enter Text:
Sumedh Naik
Select Text Effect:
Uppercase
Select Font Size:
43
Text Color
Background Color
SUMEDH NAIK



When the user clicks on the "Text Color" button, a color dialog box appears, allowing them to select the font color for the inputted text, as shown below.


Colour
Basic colours:
Custom colours:
Define Custom Colours >>
OK
Cancel
Colour|Solid
Hue: 40
Sat: 240
Lum: 120
Red: 255
Green: 255
Blue: 0
Add to Custom Colours



When the user clicks on the "Background Color" button, the same color dialog box appears, allowing the user to select the background color for the text. Here, the blue color is chosen, as shown in the output above.

Armstrong Numbers Checker and Finder

Project#5

Design a Python GUI to verify and find Armstrong numbers within a given range.

Introduction:

Welcome to our new project Armstrong Number Finder! In this project, we aim to develop a user-friendly graphical user interface (GUI) that helps users identify and find Armstrong numbers effortlessly.

Armstrong numbers are special because they have a unique property: when you add up each digit raised to the power of the total number of digits, you get the original number back.

For example,

153 is an Armstrong number because 1^3 + 5^3 + 3^3 = 153.

120 is not an Armstrong number because when we cube each digit and sum them together 1^3 + 2^3 + 0^3 = 9

, we do not get the original number.

With this Tkinter application, users can input any number they wish to check for being an Armstrong number. Upon clicking the "Check" button, the application will promptly determine whether the entered number meets the Armstrong criteria. If the number is not an Armstrong number, an error messagebox will appear, providing clear feedback about the error.

But that's not all! This application goes a step further by offering assistance to users who encounter errors. When users click "OK" on the error messagebox, a new messagebox will appear with a supportive message: "I will help you find an Armstrong number." This

message aims to encourage users and guide them towards successfully finding Armstrong numbers.

Moreover, This project allows users to explore a range of numbers to search for Armstrong numbers efficiently. By entering the start and end range values and clicking the "Generate Armstrong Number" button, users can discover all Armstrong numbers within the specified range. The results will be displayed in a text area.

We'll use Tkinter's Entry widget for number input and start and end range fields, an error messagebox for displaying errors, and a button widget for checking and verifying Armstrong numbers. Text Area for displaying result. These components will ensure that users can correctly generate Armstrong numbers.

Now, let's build this helpful form together!

Code:

```
import tkinter as tk
from tkinter import messagebox
def is_armstrong(num):
    """Check if a number is an Armstrong number."""
    # It finds out how many digits the number has and saves it
    order = len(str(num))
    # INitialize sum to zero
    sum = 0
    # Create temporary variable and copy of number to it
    temp = num
    # Loop continues until temp variable reaches to 0
    while temp > 0:
        digit = temp % 10 # Get last digit of number
       sum += digit ** order # adds the last digit raised to
      #the power of the total number of digits to the total.
        temp //= 10 # Remove last digit from temp
    return num == sum # Compares the original number num with
the calculated sum.
```

```
            # If they are equal, the function returns True,
    # indicating that the number is an Armstrong number.
        # Otherwise, it returns False.
def check_armstrong():
    """Check if the entered number is Armstrong."""
    num = int(num_entry.get()) # Get number from entry widget
    if is_armstrong(num): # call function is_armstrong and check number is armstrong and not
        # If true,result shows in messagebox
        messagebox.showinfo("Armstrong Number", f"{num} is an Armstrong number!")
    else:
        # If false, error messagebox appear
        messagebox.showerror("Error", f"{num} is not an Armstrong number.")
        # Help messagebox appear to guide user
        messagebox.showinfo("Help", "I will help you to find armstrong number in the given range")
        # Call ask_range() function to open new window
        ask_range()
def ask_range():
    """Ask user for range to display Armstrong numbers."""
    range_window = tk.Toplevel(root)
    range_window.title("Enter Range")
    name_label = tk.Label(range_window, text="Armstrong Number Generator")
    name_label.grid(row=0, columnspan=2)
    name_label.config(fg="white", bg="black", font=("Helvetica", 15, "bold"))
    # Create label for entry widget
    start_label = tk.Label(range_window, text="Start:")
    start_label.grid(row=1, column=0)
    start_entry = tk.Entry(range_window)
    start_entry.grid(row=1, column=1)
```

```
    # Create label for entry widget
    end_label = tk.Label(range_window, text="End:")
    end_label.grid(row=2, column=0)
    end_entry = tk.Entry(range_window)
    end_entry.grid(row=2, column=1)
    # Create Text Area widget
    result_text = tk.Text(range_window, height=10, width=40)
    result_text.grid(row=3, columnspan=2)
    def generate_armstrong_numbers():
        """Generate Armstrong numbers within the given range."""
        try:
            start = int(start_entry.get()) # Get number from entry widget
            end = int(end_entry.get())
            armstrong_numbers = []
            for num in range(start, end + 1):
                if is_armstrong(num):
                    armstrong_numbers.append(num)
            result_text.delete("1.0", tk.END)
            if armstrong_numbers:
                for num in armstrong_numbers:
                    result_text.insert(tk.END, f"{num}\n")
            else:
                result_text.insert(tk.END, "No Armstrong numbers in the given range.")
        except ValueError:
            result_text.insert(tk.END, "Please enter valid start and end values.")
    # Create generate button
    generate_button = tk.Button(range_window, text="Generate Armstrong Numbers", command=generate_armstrong_numbers)
    generate_button.grid(row=4, columnspan=2)
# Create the main window
root = tk.Tk()
```

```
root.title("Armstrong Number Checker")
# Create Label for Project title
# Apply font style to title label
title_label = tk.Label(root, text="Armstrong Number Finder")
title_label.pack()
title_label.config(fg="white", bg="black", font=("Helvetica", 15,
"bold"))
# Create Label for Entry widget
num_label = tk.Label(root, text="Enter a number:")
num_label.pack()
# Create entry widget
num_entry = tk.Entry(root)
num_entry.pack()
# Create Check button
check_button = tk.Button(root, text="Check",
command=check_armstrong)
check_button.pack()
# Create label for result
result_label = tk.Label(root, text="")
result_label.pack()
# Start the Tkinter event loop
root.mainloop()
```

Output:

After running the provided code, you'll see a graphical user interface (GUI) window displayed on your screen.

Number Input Field: At the center of the GUI, users will find a text input field where they can enter the number they want to check for being an Armstrong number.

Check Button: Beneath to the input field, there will be a "Check" button. Users can click this button to verify whether the entered number is an Armstrong number.

Error Messagebox: If the entered number is not an Armstrong number, an error messagebox will appear, providing clear feedback about the error. It will prompt users to acknowledge the error by clicking "OK".

Guidance Messagebox: Upon clicking "OK" on the error messagebox, a new messagebox will appear with a message such as "I will help you find an Armstrong number." This message aims to reassure users and guide them towards finding Armstrong numbers.

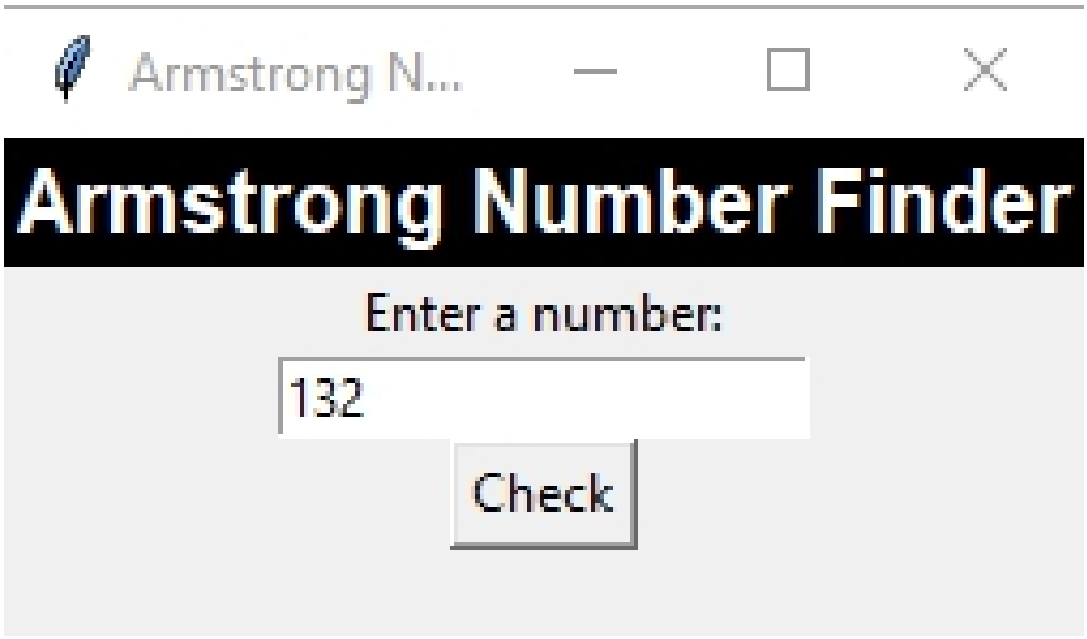
Input Range Window: When users click "OK" on the guidance messagebox, a new window will open. This window will allow users to enter the start and end range values for searching Armstrong numbers within a range.

Entry widget for Start and End Range Fields: In the input range window, users will find text input fields where they can enter the start and end range values.

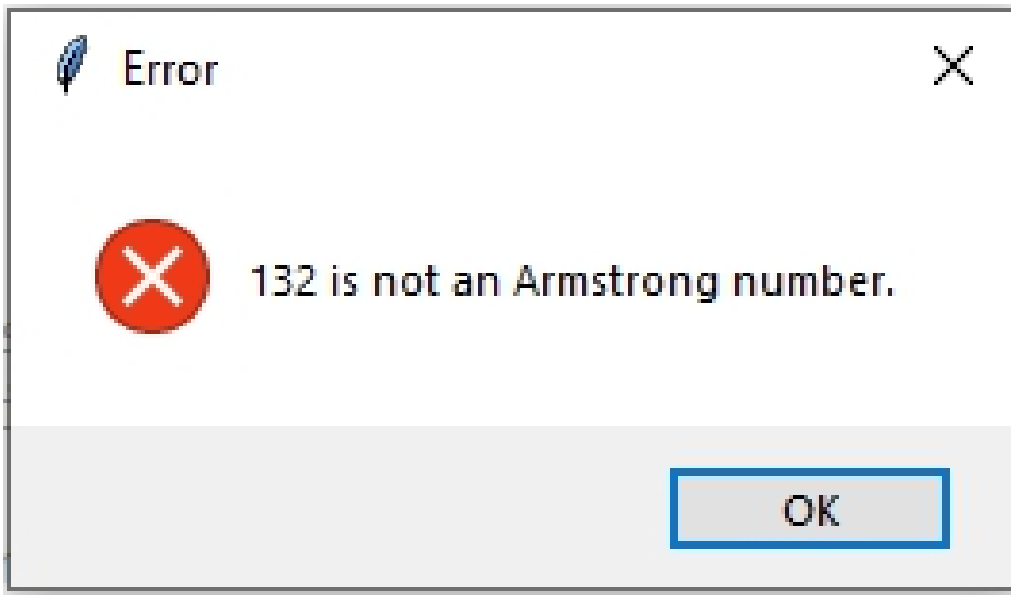
Find Button: Below the input fields, there will be a "Find" button. Users can click this button to initiate the search for Armstrong numbers within the specified range.

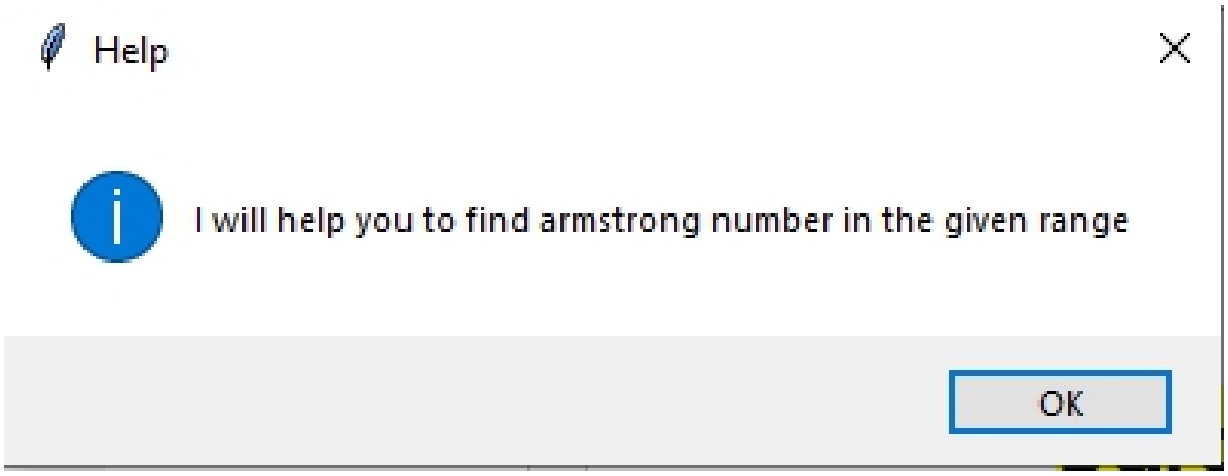
Result Text Area: After clicking the "Find" button, all

Armstrong numbers found within the specified range will be displayed in a text area below the input fields.



When a user enters the number 132 and clicks on the Check button, the following error message is displayed:

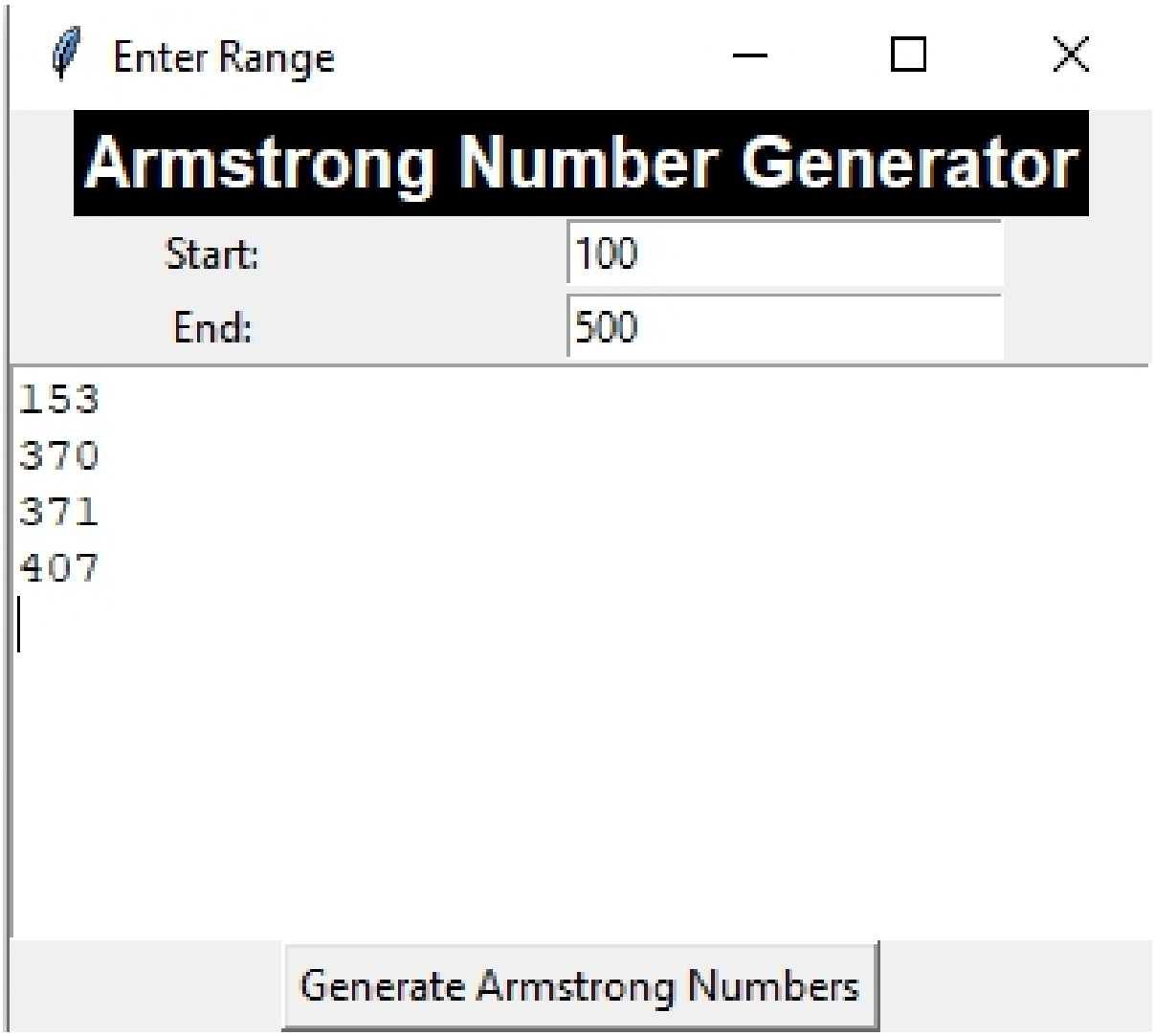




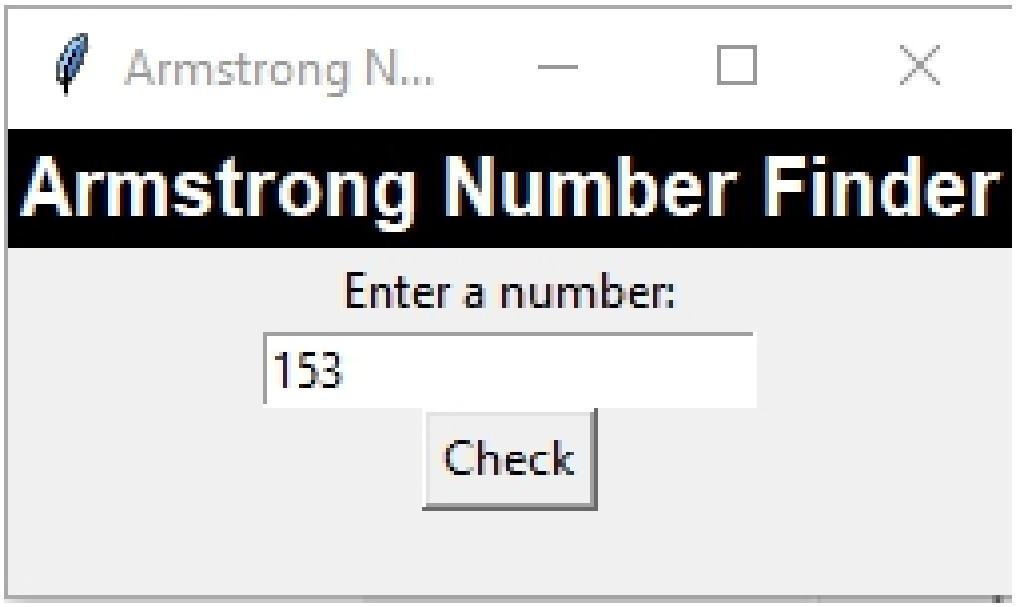
After clicking the OK button, the above help message box is displayed on the screen.

After clicking the OK button, a new range window will appear as depicted below. Before we proceed, let's manually check why 132 is not Armstrong number:

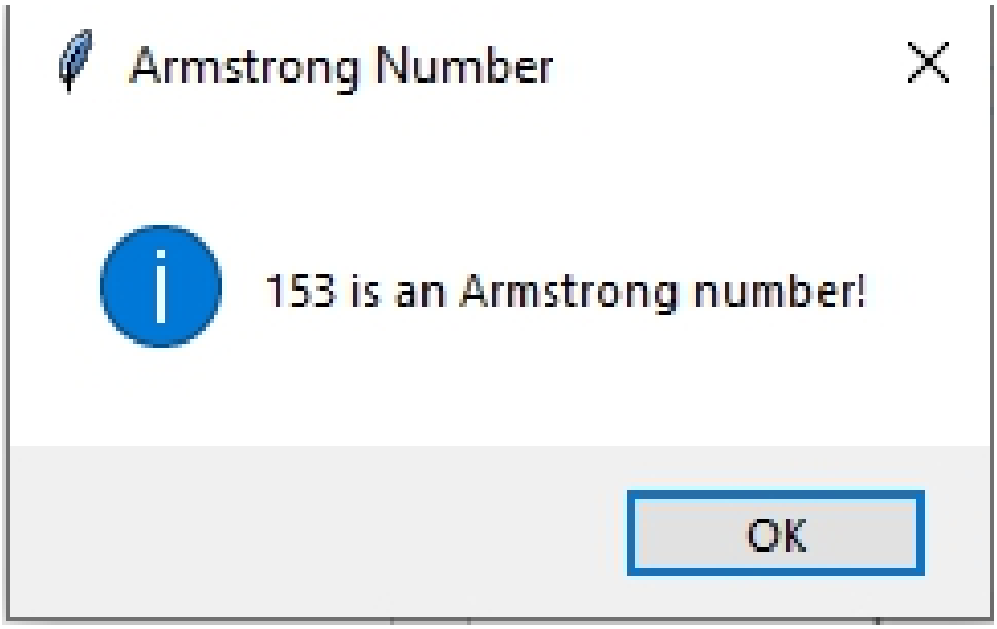
132 is not an Armstrong number because when we cube each digit and sum them together 1^3 + 3^3 + 2^3 = 1 + 27+8 = 36, we do not get the original number.



When the user inputs the start and end numbers and clicks on the 'Generate Armstrong Numbers' button, the application generates a list of Armstrong numbers within the specified range. This list is then displayed in the text area, as demonstrated in the interface above. Here,The user entered 100 and 500 as the start and end numbers. After clicking, 153, 370, 371, and 407 were displayed in the text area as shown.



Upon entering the number 153, which is an Armstrong number, and clicking the Check button, a message box appears with the following content.



Barcode Scanner

Project #6

Design a Barcode Scaner application using Tkinter in Python

Introduction:

Welcome to our Barcode Scanner application! This project aims to provide a simple solution for extracting information from barcode images. With just a few clicks, users can select an image containing a barcode, and our application will decode the barcode data, displaying it conveniently for further use. Using Tkinter, a Python library for creating graphical user interfaces, we've designed a simple interface that allows users to browse their file system and select the desired barcode image. Once selected, the application utilizes the powerful pyzbar library to decode the barcode data embedded within the image.

The workflow is simple: upon pressing the "Browse" button, a file dialog box appears, enabling users to navigate to the location of their barcode image. After selecting the image, our application processes it, extracts the barcode data, and promptly displays it on a label situated conveniently below the browse button.

If you're interested in learning how to generate barcode images, you can refer to my book, titled 'Python Tkinter 35 Mini Projects.' I've included the Barcode Generator project in detail, along with the complete source code and explanations.

Lets build this powerful tool together.

Requirement:

We will use different TKinter modules to develop this simple GUI form. They are as follows:

Modules :

Tkinter: Tkinter is a standard GUI (Graphical User Interface) toolkit for Python. It allows the creation of windows, labels, buttons, and other GUI elements. Tkinter is included with Python by default, so no additional installation is necessary as we discuss about it in first project.

In this project, we will use the filedialog module to open and select barcode images, and the messagebox module to display error messages if a barcode image file is not found. To use these modules, we need to import them from the tkinter library. Here's how we do it:

from tkinter import filedialog, messagebox

This statement imports both filedialog and messagebox modules from tkinter, enabling us to utilize file dialogs and message boxes in our application.

Pyzbar: Pyzbar is a Python library for decoding barcodes from images. It supports various barcode formats such as QR codes, UPC codes, and EAN codes.

You can install Pyzbar using pip:

pip install pyzbar

PIL (Python Imaging Library) or Pillow: PIL or Pillow is a Python library for opening, manipulating, and saving many different image file formats.

You can install Pillow using pip:

pip install pillow

In this project, we will be working with image manipulation. To facilitate this, we'll need to import the Image module from the PIL library. We can achieve this with the following code:

from PIL import Image

To extract barcode data from images, we'll need to scan and decode them. This requires importing the decode function from pyzbar. We can accomplish this with the following statement:

from pyzbar.pyzbar import decode

Code:

```
import tkinter as tk
from tkinter import filedialog, messagebox
from PIL import Image
from pyzbar.pyzbar import decode

def browse_file():
    # Open file dialog box to select image
    filepath = filedialog.askopenfilename
            (filetypes=[("Image Files", "*.png;*.jpg;*.jpeg")])
    # Check image file is open or not
    if filepath:
        # Attempt to open image file
        try:
            # If it is open, extract data from image using decode function
            barcode_data =decode(Image.open(filepath))
            # Check data is properly decoded or not
            if barcode_data:
                # If True, it display extracted data on label
                result_label.config(text="Barcode: " + barcode_data[0].data.decode('utf-8'))
            else:
                result_label.config(text="No barcode found in the image.")
        except Exception as e:
            # Error message is display when it falied to open file
            messagebox.showerror("Error", "Failed to open image file: " + str(e))
```

```
# Create Tkinter window
root = tk.Tk()
root.title("Barcode Scanner")

# Create Label for Project title
# Apply font style to title label
title_label = tk.Label(root, text="Barcode Scanner")
title_label.pack(pady=10)
title_label.config(fg="white", bg="Dark Red", font=("Helvetica", 15,
"bold"))

# Create GUI elements
label = tk.Label(root, text="Select Barcode Image:")
label.pack()

# Create Browse button to select input barcode image
browse_button = tk.Button(root, text="Browse",
command=browse_file)
browse_button.pack(pady=5)

# Create label to display result
result_label = tk.Label(root, text="")
result_label.pack()

# Run the Tkinter event loop
root.mainloop()
```

Output :

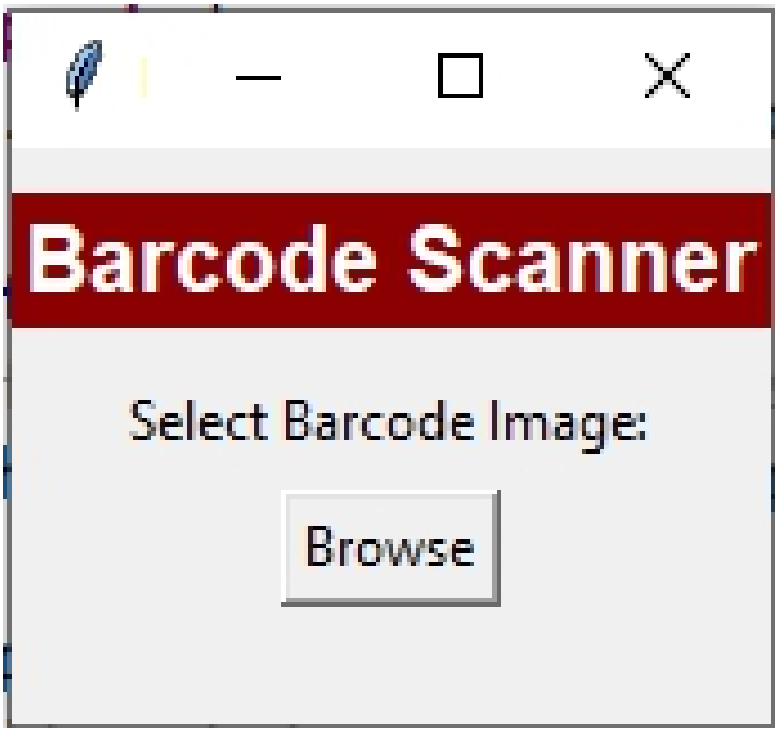
After executing the code for the Barcode Scanner project in a GUI, you will see a window with the following components:

Title: At the top of GUI, there is a label with text “ Barcode Scanner”,providing an indication of the purpose of the application.

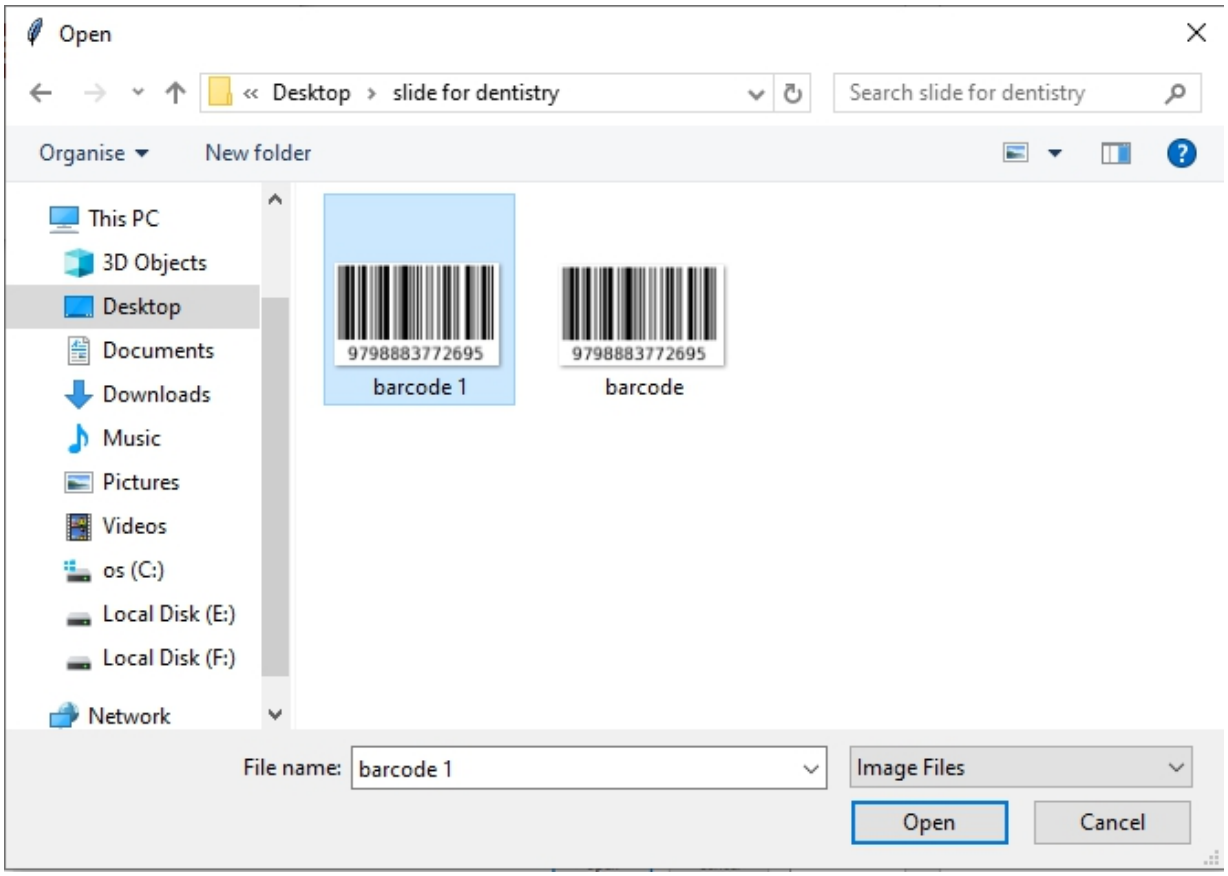
Label: A label with the text "Select Barcode Image:". This label prompts the user to select a barcode image.

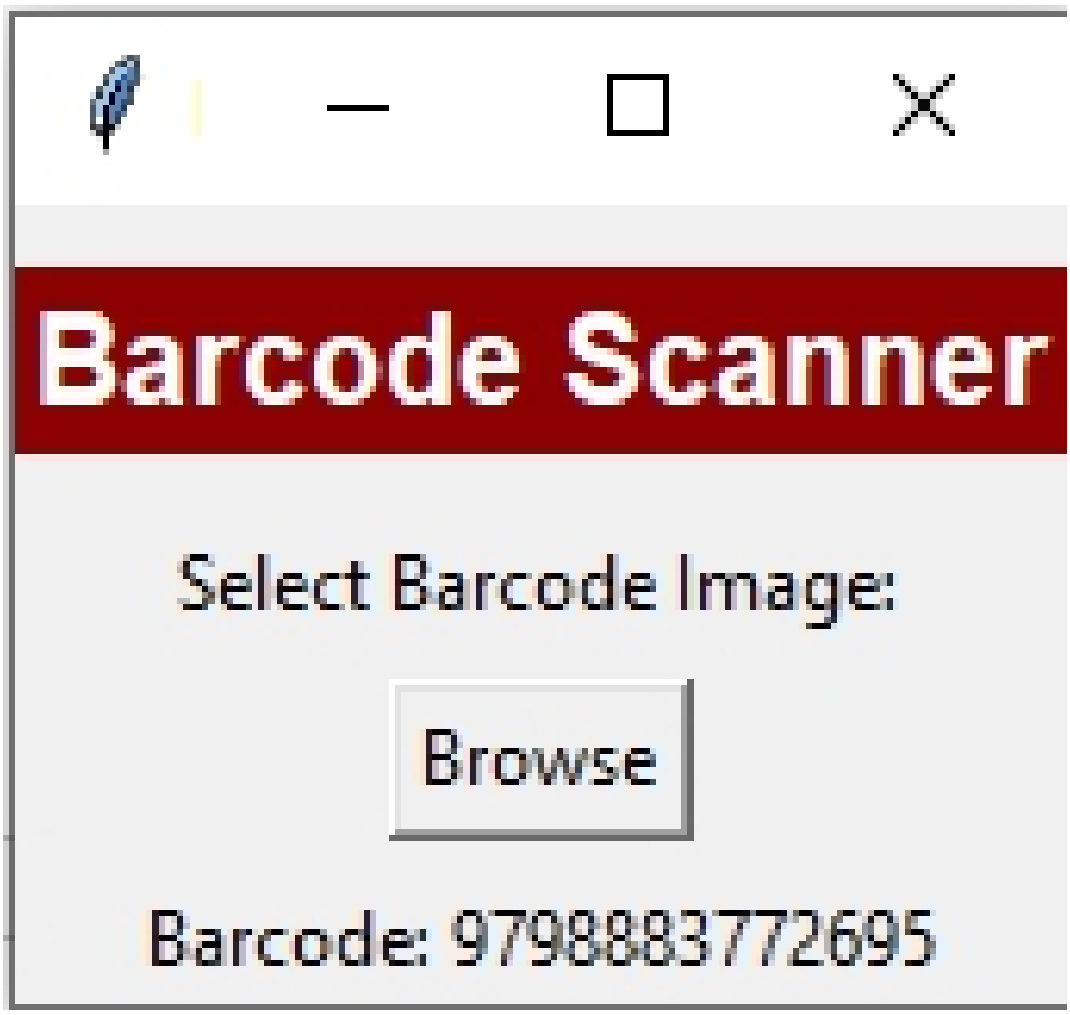
Browse Button: Besides the Label ,there is a button labeled "Browse". Clicking this button will open a file dialog box, allowing the user to navigate their file system and select the desired barcode image file.

Result Label: Besides the Browse button, there is result label which is Initially empty. After selecting a barcode image and decoding it, this label will display the extracted barcode data.



After clicking the Browse button, following open dialog box is displayed on your screen. Select the barcode image file from the folder and click on the open button as shown below


Open
« Desktop › slide for dentistry
Search slide for dentistry
Organise
New folder
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
os (C:)
Local Disk (E:)
Local Disk (F:)
Network
9798883772695
barcode 1
9798883772695
barcode
File name:
barcode 1
Image Files
Open
Cancel



Upon successful extraction and decoding of barcode information from an image file, the barcode data is displayed on the main window, as depicted above.

Offline Image DPI Convertor

Project #7

Design Offline Image DPI convertor project using Python Tkinter

Introduction:

Welcome to our Offline Image DPI Converter app! This tool helps you improve the quality of your pictures without needing the internet. It's super easy to use. Just click the "Browse" button to pick the image you want to make better. Then, choose how sharp you want it to be by selecting 100, 200, 300, or 400 DPI.

Once you've picked your DPI, hit the "Increase Resolution" button, and voilà! Your image will be clearer and sharper, ready for printing or sharing. Whether you're a photographer or just someone who loves taking pictures, this app makes it simple to enhance your images.

To build the GUI for this project, we'll use Tkinter's OptionButton widget to select the DPI for sharper images, and a Button widget to open the file dialog box for image selection and display high-resolution images. These components will ensure users can generate clear and sharp images as desired. Let's build this helpful form and make your photos look their best together !"

Code :

```
import tkinter as tk
from tkinter import filedialog
from PIL import Image, ImageTk

def increase_resolution():
    try:
        # Get the selected image file path
        image_path = file_path.get()
```

```
        # Open the image file
        image = Image.open(image_path)

        # Get the desired DPI from user selection
        dpi = int(dpi_var.get())

        # Update the DPI metadata
        image.info["dpi"] = (dpi, dpi)

        # Save the image with updated DPI
        image.save("resized_image.jpg", dpi=(dpi, dpi))

        # Display the resized image in a new window
        display_resized_image(image)

        # Update the status label
        status_label.config
            (text="Image resolution increased successfully!")
    except Exception as e:
        # Show error if any
        Error_label.config(text="Error: " + str(e))

def display_resized_image(image):
    # Create a new window to display the resized image
    window = tk.Toplevel(root)
    window.title("Resized Image")

    # Convert PIL image to Tkinter PhotoImage
    photo = ImageTk.PhotoImage(image)

    # Display the resized image in a label
    label = tk.Label(window, image=photo)
    label.image = photo  # Keep a reference to avoid garbage collection
    label.pack()

def browse_file():
    # Open a file dialog to select an image file
    file_path.set(filedialog.askopenfilename())
    # Update status label
```

```
    status_label.config(text="Image is selected for   conversion successfully!")

# Create the main window
root = tk.Tk()
root.title("Image Resolution Increaser")

# Create Label for Project title
# Apply font style to title label
title_label = tk.Label(root, text="Offline DPI Convertor")
title_label.grid (row=0, column=0, columnspan=2, pady=10)
title_label.config(fg="white", bg="Dark Red", font=("Helvetica", 15, "bold"))

# Variables
file_path = tk.StringVar()
dpi_var = tk.StringVar()

# Widgets
# Create label
label_select = tk.Label(root, text="Select Image:")
label_select.grid(row=1, column=0, padx=5, pady=5)
# Create browse button
btn_browse = tk.Button(root, text="Browse", command=browse_file)
btn_browse.grid(row=1, column=1, padx=5, pady=5)
# Create label fro DPI options
label_dpi = tk.Label(root, text="Select DPI:")
label_dpi.grid(row=2, column=0, padx=5, pady=5)

# Define DPI options for image resolution settings
dpi_options = [100, 200, 300, 500]

# Initialize row index
row_index = 2

# Iterate over each DPI option and create a Radiobutton for each
for dpi_option in dpi_options:
    # Create Radiobutton widget with DPI option as text label
```

```
    rb_dpi = tk.Radiobutton(root, text=str(dpi_option),
variable=dpi_var, value=str(dpi_option))
    # Place Radiobutton in the GUI grid layout with padding
    rb_dpi.grid(row=row_index, column=1, padx=5, pady=5)
    # Increment row index for the next iteration
    row_index += 1
# Row index for placing additional widgets
row_index = 6  # You can change this value as needed

# Create Increase Resolution button
btn_increase_resolution = tk.Button(root, text="Increase Resolution",
command=increase_resolution)
btn_increase_resolution.grid(row=row_index, columnspan=2,
padx=5, pady=5)

# Create status label
status_label = tk.Label(root, text="", fg="green")
status_label.grid(row=row_index + 1, columnspan=2, padx=5,
pady=5)

# Create error label
error_label = tk.Label(root, text="", fg="red")
error_label.grid(row=row_index + 2, columnspan=2, padx=5,
pady=5)
# Start the main event loop
root.mainloop()
```

Output:

After executing the provided code for the Offline Image DPI Converter, the GUI will display the following components:

Title : At the top of the window, you'll see the title of the application, which says 'Offline DPI Converter'."

Label : Beneath the form title, there is a instructional label that reads : Select image.

Browse Button: Positioned within the interface, you will find a button labeled "Browse". Clicking this button will open a file dialog,

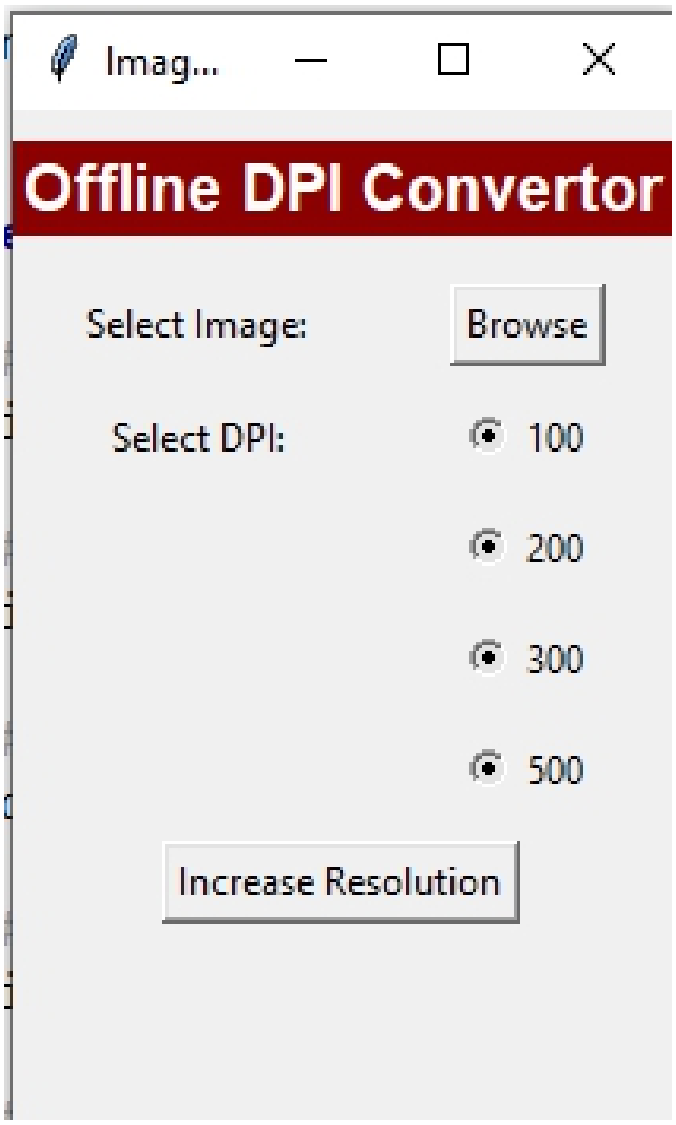
allowing you to select the image file you wish to enhance.

DPI Options: Beneath Browse button,you will find a set of radio buttons presenting predefined DPI options, such as 100, 200, 300, 400 and 500 DPI. You can choose the desired DPI setting from these options.

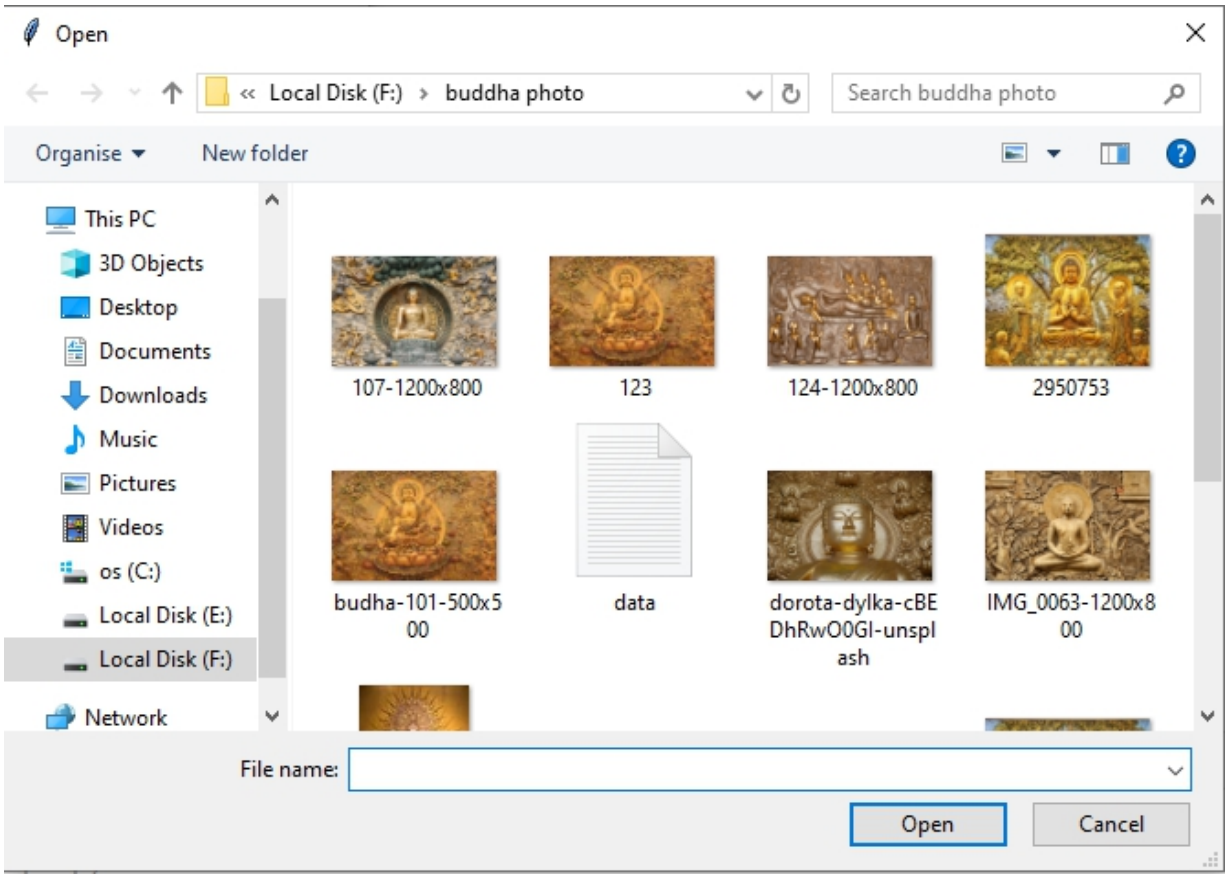
Increase Resolution Button: Below the DPI options, there's a button called 'Increase Resolution.' When you click this button, it starts the process of converting the DPI based on the option you selected.

Status Label : At the bottom of the window, you'll find a status bar. It shows messages to help you during the

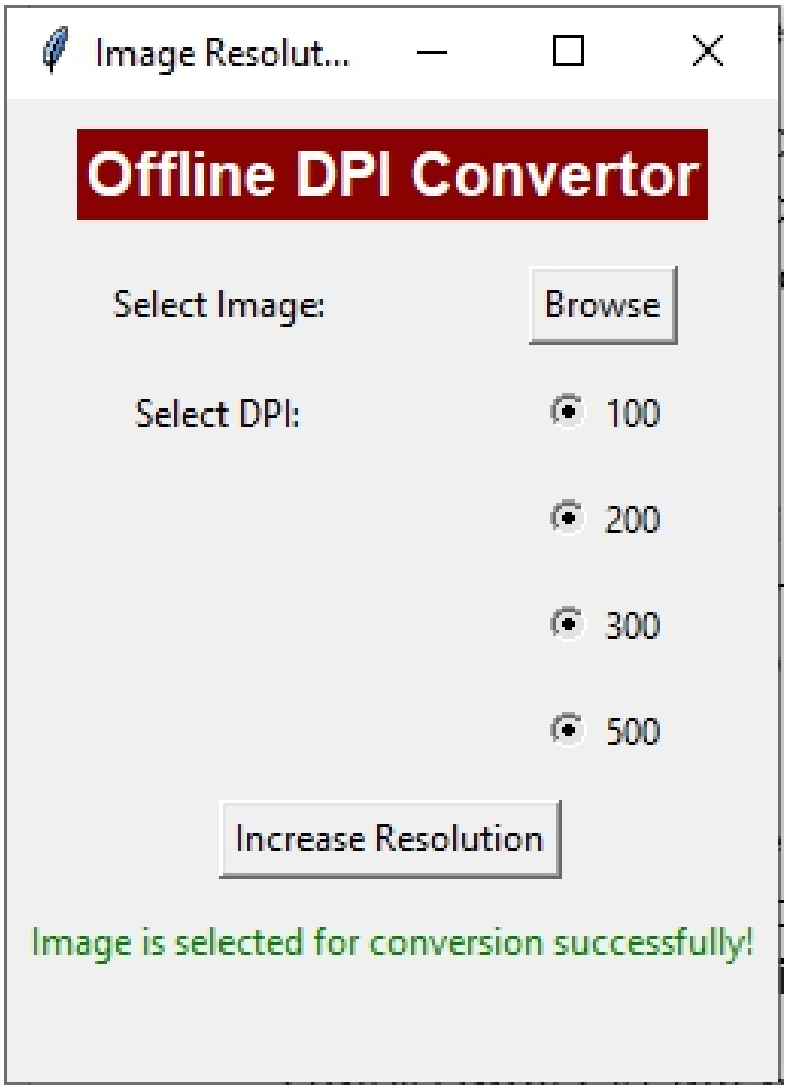
enhancement process. This can include updates about importing images, progress on DPI conversion, or notifications about any errors.



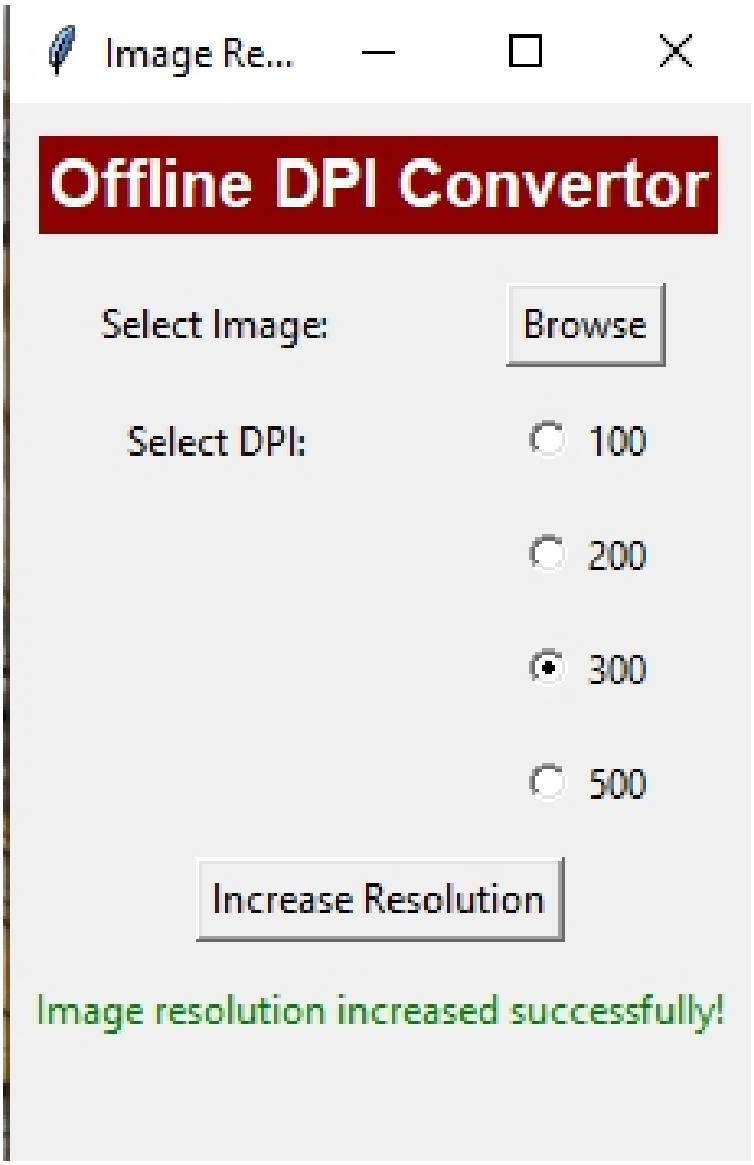
This is the main window of the project where users can select any image to enhance its clarity and sharpness by choosing DPI values.



After clicking the Browse button, above open dialog box is displayed on your screen. Select the image file from the folder and click on the open button.



After successfully uploading the image for conversion, the status label is updated with the message 'Image is successfully selected for conversion' as shown above



After selecting a DPI from the option button and clicking on the 'Increase Resolution' button, the status label is updated with the message 'Image resolution increased successfully,' and the resultant image is automatically displayed as shown below


Resized Image

Emoji Printer

Project #8

Develop a Graphical User Interface (GUI) application using Tkinter in Python to print Emojis alongwith text.

Introduction:

Welcome to the Emoji Printer project! Here, you can make your messages more fun by adding emojis. It's easy to do: just type your message in the box provided. Underneath, you'll find a menu with lots of emojis to choose from. Pick one you like, and click the "Add Emoji" button. Your chosen emoji will appear right where you were typing! Once you're happy with your message, click the "Print Message" button. Your message will pop up on the screen, but with a twist – it'll be in bold font and filled with your chosen emojis. It's a fun way to make your messages stand out!

To build the GUI for this project, we'll use Tkinter's Entry widget for entering messages, a dropdown menu for selecting emoji symbols, a button widget for adding emojis to the text and displaying the text message on the window, and a Label widget to show the message with emojis on the screen. Lets try to implement this !

Code:

```
import tkinter as tk
from tkinter import ttk
# Function to add emoji to current selection
def add_emoji():
    # Retrieve the selected emoji name from a combo box.
    selected_emoji_name = emoji_combobox.get()
    # Find the emoji's symbol based on its name, or say it's not found."
    selected_emoji =
emoji_name_to_unicode.get(selected_emoji_name, "Emoji not
found!")
```

```
    # Put the selected emoji symbol where the cursor is in the text box.
    text_entry.insert(tk.INSERT, selected_emoji)  # Insert emoji at current cursor position

# Define a function named print_emoji.
def print_emoji():
    # Get the text from the text box, removing any extra spaces or lines.
    printed_text = text_entry.get("1.0", tk.END).strip()
    # Update the output label to display the entered text.
    output_label.config(text=printed_text)

# Create a window for the Emoji Writer application.
root = tk.Tk()

# Set the title of the window as 'Emoji Writer'.
root.title("Emoji Writer")

# Define categories of emojis with their respective names.
emoji_categories = {
    "Smileys & Emotion": ["grinning", "grin", "joy", "rofl",   "sob",
                          "neutral_face", "smirk", "heart_eyes",
                          "kissing_heart", "face_with_tears_of_joy",
                          "rolling_on_the_floor_laughing"],
    "Animals & Nature": ["dog", "cat", "mouse", "hamster",
                         "fox_face", "bear","panda_face", "koala",
"rabbit","tiger", "lion_face",
                         "horse", "unicorn"],
    "Flowers & Fruits" : ["rose","sunflower","tulip","seedling",
                          "strawberry","melon","cherry","peach",
                          "mango","pineapple","coconut","kiwi","tomato"]
}

# Create a dictionary mapping emoji names to their Unicode representations.
emoji_name_to_unicode = {
```

```
    "grinning": "\U0001F600","grin" : "\U0001F601","joy":
"\U0001F602",  "rofl": "\U0001F923","sob":
"\U0001F62D","neutral_face":"\U0001F610",
    "smirk": "\U0001F60F","heart_eyes":
"\U0001F60D","kissing_heart": "\U0001F618",
    "face_with_tears_of_joy":
"\U0001F605","rolling_on_the_floor_laughing": "\U0001F923", "dog":
"\U0001F436","cat": "\U0001F431","mouse":
"\U0001F42D","hamster": "\U0001F439",
    "fox_face": "\U0001F981","bear": "\U0001F43B","panda_face":
"\U0001F43C","koala": "\U0001F428",
    "rabbit": "\U0001F430","tiger": "\U0001F42F","lion_face":
"\U0001F981","horse": "\U0001F434","unicorn": "\U0001F984",
    "rose": "\U0001F339","sunflower": "\U0001F33B","tulip":
"\U0001F337","seedling": "\U0001F331",
    "strawberry": "\U0001F353","melon": "\U0001F348","cherry":
"\U0001F352","peach": "\U0001F351",
    "mango": "\U0001F353","pineapple": "\U0001F348","coconut":
"\U0001F352","kiwi": "\U0001F351","tomato":"\U0001F345"
}
```

 # Create label for title

```
title_label = tk.Label(root, text="Emoji Printer")
title_label.pack()
title_label.config(fg="yellow", bg="blue", font=("Calibri", 30, "bold"))
```

 # Create label

```
text_label = tk.Label(root, text="Write your Message :")
text_label.pack()
```

 # Text Entry

```
text_entry = tk.Text(root, height=3, width=40)
text_entry.pack(pady=10)
```

 # Emoji Selection

```
emoji_label = ttk.Label(root, text="Select Emoji:")
```

```
emoji_label.pack()

# Create a Combobox for emoji selection
# Generate a list of emoji names for the dropdown menu.
# values = list(emoji_name_to_unicode.keys()), It generate list of emoji names for combobox
# It shows available emoji names and is set to read-only.
emoji_combobox = ttk.Combobox(root, values=list(emoji_name_to_unicode.keys()), state="readonly")
emoji_combobox.pack()

#  Create a Add Emoji Button
select_button = ttk.Button(root, text="Add Emoji ", command=add_emoji)
select_button.pack(pady=10)

# Create a Print Message Button
print_button = ttk.Button(root, text="Print Message", command=print_emoji)
print_button.pack(pady=10)

# Create a Label to display selected emoji
output_label = ttk.Label(root, text="", font=("Helvetica",20), wraplength=400)
output_label.pack(pady=10)

root.mainloop()
```

Output:

After executing provided code, you will see,

Text Box: At the center of the interface, users will find a text box where they can type their messages.

Emoji Selection: Directly beneath the text box, users will find a combo box populated with a delightful array of emojis. From smiley faces to hearts, users can choose from a large selection of emojis to add them to their messages.

Add Emoji Button: Beneath the text Area,there is a button labeled "Add Emoji" allow users to integrate emojis into their messages. With just a click, users can easily insert their chosen emoji into the text box.

Print Message Button: Below the text box and emoji selection, a button labeled "Print Message" allow users to print out their message,including emojis, in a big font.

Label : Below the "Print Message" button, there's a label initially displaying nothing. After clicking the "Print Message" button, the label updates to display a message along with emojis.

After running the provided code, you will see following GUI

This is main GUI

After writing the text and selecting emojis from the dropdown menu, and then clicking on the 'Add Emoji' button, the emojis are added to the text as shown above.


Emoji Writer
Emoji Printer
Write your Message :
My son's name is Sumedh Naik😍
I love him very much😘
Select Emoji:
kissing_heart
Add Emoji
Print Message
My son's name is Sumedh Naik😍
I love him very much😘

After clicking the 'Print Message' button, the text from the textbox will appear in the main window, just as shown above.

Palindrome number Tester

Project #9

Design simple Palindrome Number Tester GUI using Tkinter and Python

Introduction:

Welcome to the Palindrome Tester project! The goal of this project is to help you find palindromic numbers easily. Let's talk about palindrome numbers.

Palindromic numbers are those that read the same forwards and backwards. In other words, the number remains unchanged when reversed. If a number doesn't match its reverse, it's not a palindrome number.

Example : 121 is palindrome number because 121 reads as 121 from left to right and from right to left.

However, -121 is not considered a palindrome number. While it reads as -121 from left to right, when read from right to left, it becomes 121-. Hence, it fails to meet the definition of a palindrome. Now, you know about palindrome number. Let’s see what is in our project.

In this project, you can pick a range of numbers by typing in the smallest and largest values you want to check. There are three options to choose from:

Palindromes Divisible by 5: Discover numbers in your chosen range that are palindromes and can be divided by 5.

Square Palindromes: Find numbers in your range whose square roots are whole numbers.

All Palindromes: See all palindromic numbers in your range, no matter what.

After picking an option, just click the "Get Palindromes" button to see the numbers. They'll show up in a box that you can scroll through. Let's start exploring palindromic numbers together!

Requirements :

Module:

Tkinter: We will use Tkinter's Entry widget for inputting minimum and maximum values, option buttons to choose the type of palindrome, a Button widget to initiate the generation of palindrome numbers, and a Text widget for displaying the results.

Scrolledtext : In Python, the ScrolledText widget from the tkinter library is used to create a text area with a vertical scroll bar. Here's how you can import it:

from tkinter import scrolledtext

The statement **from tkinter import scrolledtext** in Python means that you are importing the scrolledtext module from the tkinter library. This module provides the ScrolledText widget, which is a text widget with a built-in vertical scroll bar. By importing scrolledtext, you can use the ScrolledText widget in your application to create a text area with automatic vertical scrolling. This is useful for text areas where the content might exceed the visible area, allowing users to scroll through the text.

In this project, we will use the "scrolledtext" module to show all palindrome numbers in a given range. The scrollbar helps users to see the numbers easily, especially if there are many.

Code :

```
import tkinter as tk

from tkinter import scrolledtext
```

```
# Define a function called is_palindrome that takes a number n as input.
def is_palindrome(n):
    # Convert the number n to a string and check if it's equal to its reverse.
    return str(n) == str(n)[::-1]  # Return True if the string representation of n is a palindrome, otherwise return False.

# Define a function called get_palindromes that takes three arguments: min_val, max_val, and option.
def get_palindromes(min_val, max_val, option):
    # Create an empty list called palindromes to store palindrome numbers.
    palindromes = []

    # Check the value of the option argument.
    if option == 1:  # Palindrome divisible by 5
        # If option is 1 (Palindrome divisible by 5), iterate through the range of numbers from min_val to max_val.
        for i in range(min_val, max_val + 1):
            # If the number is a palindrome and divisible by 5, add it to the palindromes list.
            if is_palindrome(i) and i % 5 == 0:
                palindromes.append(i)
    elif option == 2:  # Square palindrome
```

```
        # If option is 2 (Square palindrome), iterate through the range of numbers from min_val to max_val.
        for i in range(min_val, max_val + 1):
            # If the number is a palindrome and its square root is an integer, add it to the palindromes list.
            if is_palindrome(i) and (i ** 0.5).is_integer():
                palindromes.append(i)
    else:  # All palindromes
        # If option is neither 1 nor 2, iterate through the range of numbers from min_val to max_val.
        for i in range(min_val, max_val + 1):
            # If the number is a palindrome, add it to the palindromes list.
            if is_palindrome(i):
                palindromes.append(i)

    # Return the list of palindromes.
    return palindromes

# Define a function called display_palindromes.
def display_palindromes():
    # Retrieve the minimum value from the min_entry widget and convert it to an integer.
    min_val = int(min_entry.get())
```

```
    # Retrieve the maximum value from the max_entry widget and convert it to an integer.
    max_val = int(max_entry.get())
    # Retrieve the selected option from the option_var variable.
    option = option_var.get()
    # Call the get_palindromes function with the minimum value, maximum value, and selected option, and store the result in the palindromes variable.
    palindromes = get_palindromes(min_val, max_val, option)
    # Clear the text displayed in the result_text widget.
    result_text.delete(1.0, tk.END)
    # Iterate through each palindrome in the palindromes list.
    for palindrome in palindromes:
        # Insert each palindrome followed by a newline character into the result_text widget.
        result_text.insert(tk.END, str(palindrome) + '\n')
# Create main window
root = tk.Tk()
root.title("Palindrome Checker")
# Create title label
title_label = tk.Label(root, text="Palindrome Checker and Generator")
title_label.grid(row=0, column=0, columnspan=3, pady=10)
```

```
title_label.config(fg="white", bg="Dark Green", font=("Arial", 18, "bold"))

# Create input fields

min_label = tk.Label(root, text="Min Value:")

min_label.grid(row=1, column=0)

min_entry = tk.Entry(root)

min_entry.grid(row=1, column=1)

max_label = tk.Label(root, text="Max Value:")

max_label.grid(row=2, column=0)

max_entry = tk.Entry(root)

max_entry.grid(row=2, column=1, pady=5)

# Create frame

option_frame = tk.Frame(root)

option_frame.grid(row=3, column=0, columnspan=3, pady=5)

# Create option buttons

option_label = tk.Label(option_frame, text="Options :")

option_label.grid(row=0, column=0)

option_label.config(font=("TkDefaultFont", 10, "bold"))

option_var = tk.IntVar()

option_var.set(0)

option1= tk.Radiobutton(option_frame, text="Palindrome Divisible by 5", variable=option_var, value=1)
```

```
option1.grid(row=1, column=0)

option2 = tk.Radiobutton(option_frame, text="Square Palindrome", variable=option_var, value=2)

option2.grid(row=1, column=1)

option3 = tk.Radiobutton(option_frame, text="All Palindromes", variable=option_var, value=3)

option3.grid(row=1, column=2)

# Create button to get palindromes

get_palindrome_button = tk.Button(root, text="Get Palindromes", command=display_palindromes)

get_palindrome_button.grid(row=7, column=0, columnspan=3, pady=5)

# Create text widget to display results

result_text = scrolledtext.ScrolledText(root, width=50, height=10)

result_text.grid(row=8, column=0, columnspan=2)

root.mainloop()
```

Output :

After executing the provided code, you will see a GUI window titled "Palindrome Number Tester".

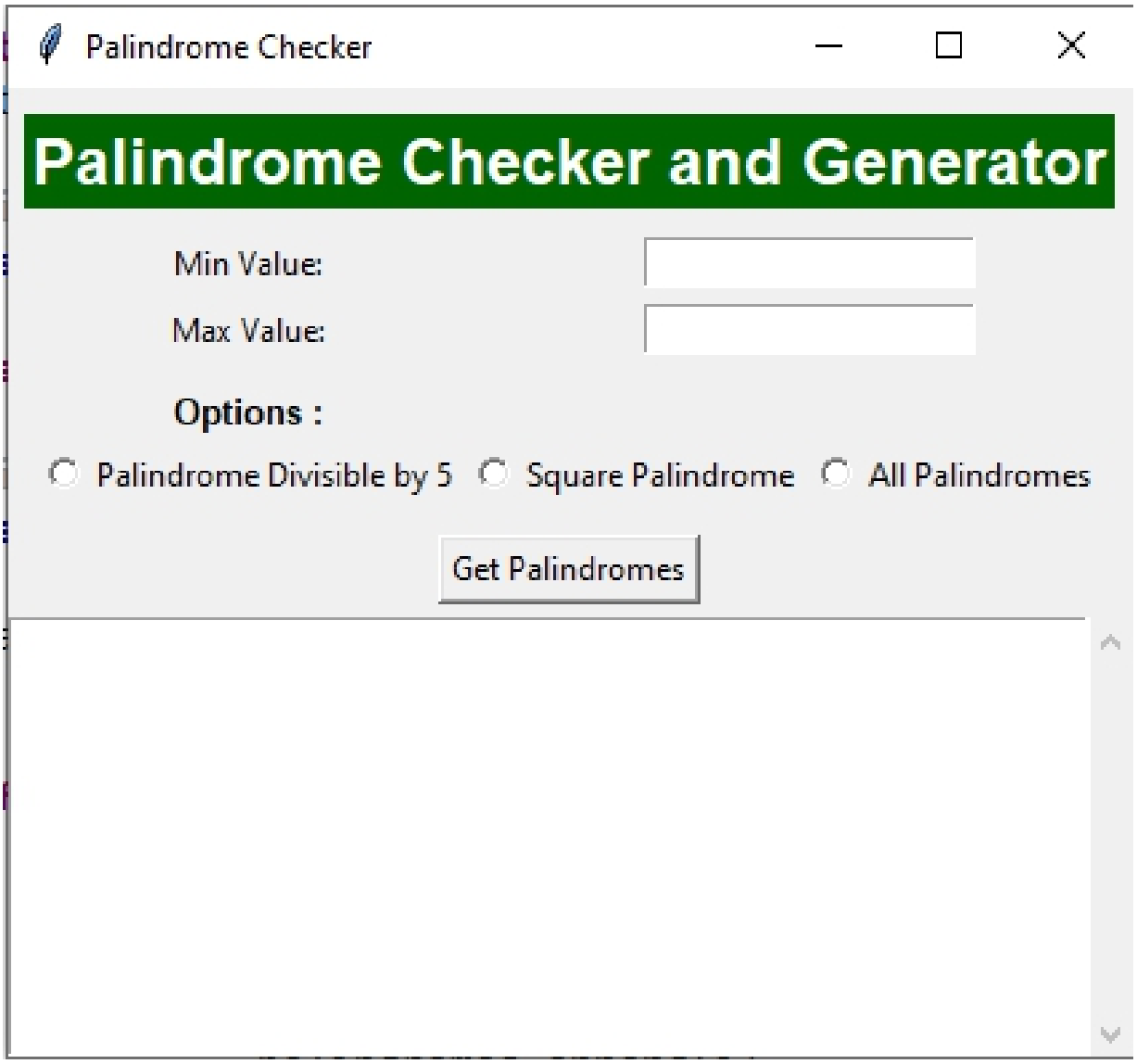
Input Fields: At the top of GUI, you will find Two entry fields labeled "Minimum Value" and "Maximum Value" where you can input the range of numbers you want to check for palindromes.

Options to Choose From: Below the input fields, you'll see three radio buttons or option buttons labeled "Palindrome Divisible by 5",

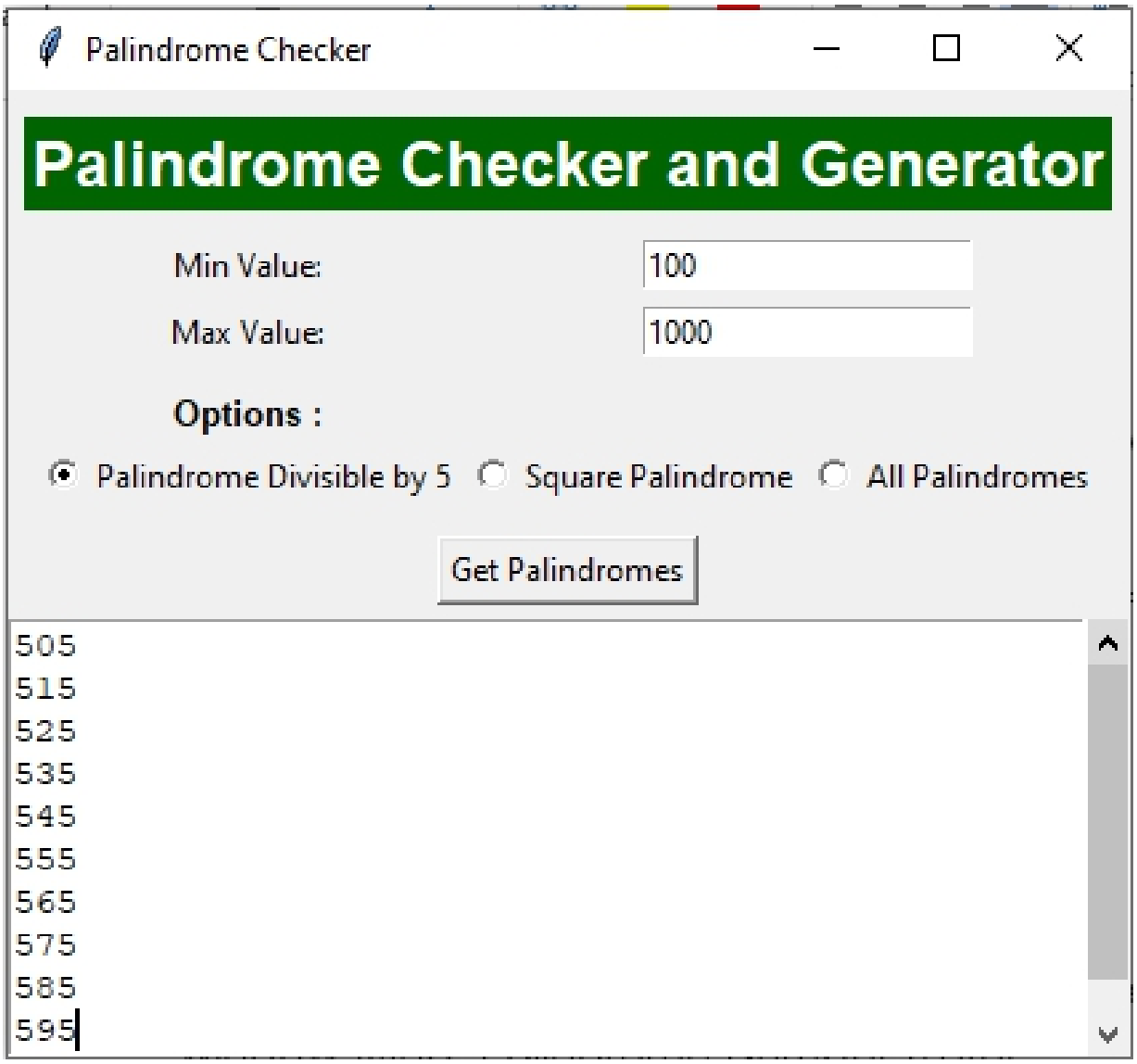
"Squared Palindromes", and "All Palindromes" to choose the type of palindromes you want to find.

"Get Palindromes" Button: After entering your desired range and selecting an option, you will see a button labeled "Get Palindromes." Clicking this button initiates the process of finding and displaying the palindrome numbers based on your chosen criteria.

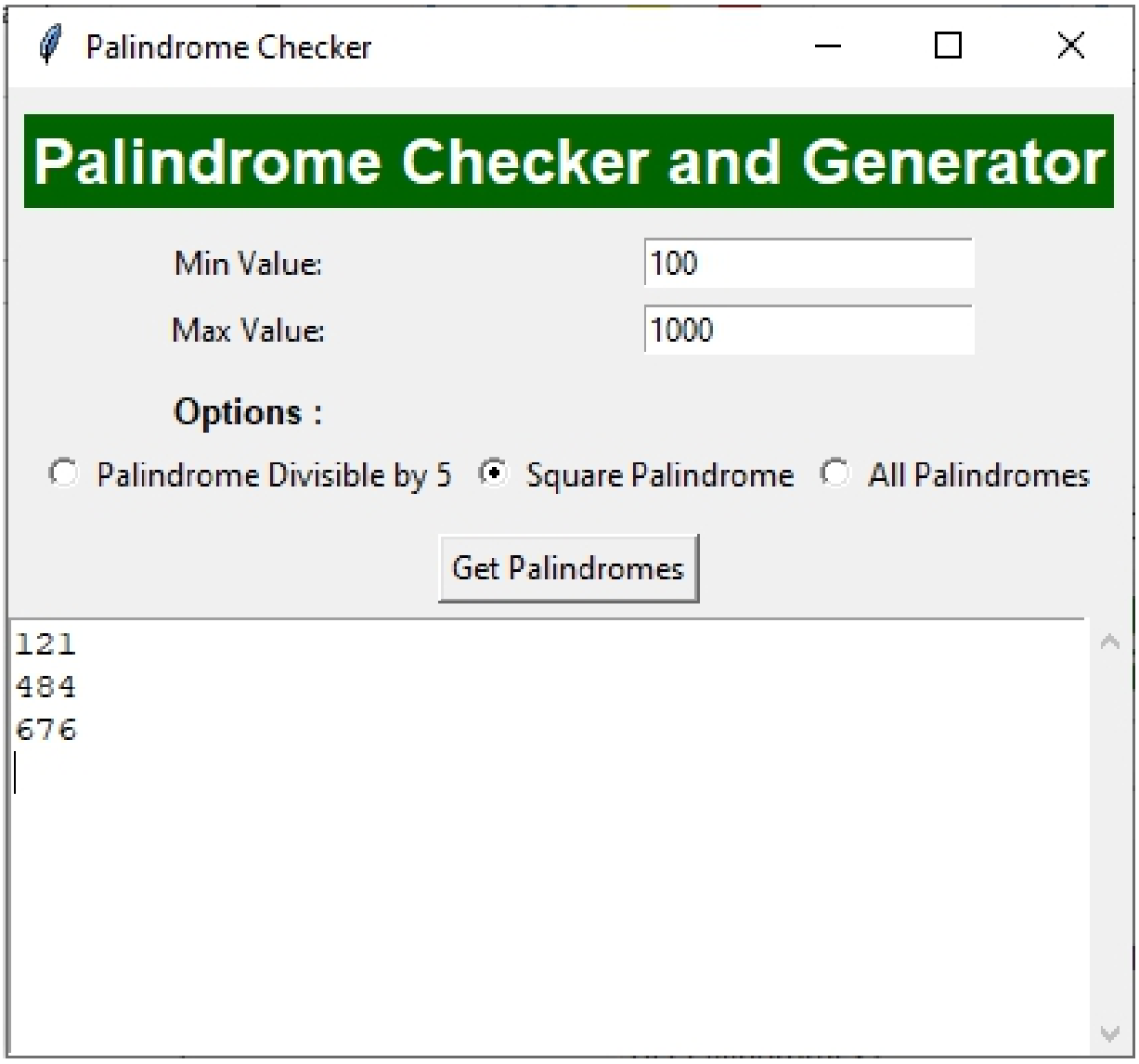
Text Display Area: Once you click the "Get Palindromes" button, all palindrome number found within the specified range according to selected option will appear in text disaply area. A vertical scrollbar may be provided if the list of palindrome numbers exceeds the display area's capacity.



This main window allows the user to enter the minimum and maximum values to find palindrome numbers. Users can select the type of palindrome number they want to generate.



When the user inputs the range from 100 to 1000 and click on Get Palindrome button, the text area displays all palindrome numbers within this range that are divisible by 5, as demonstrated above.

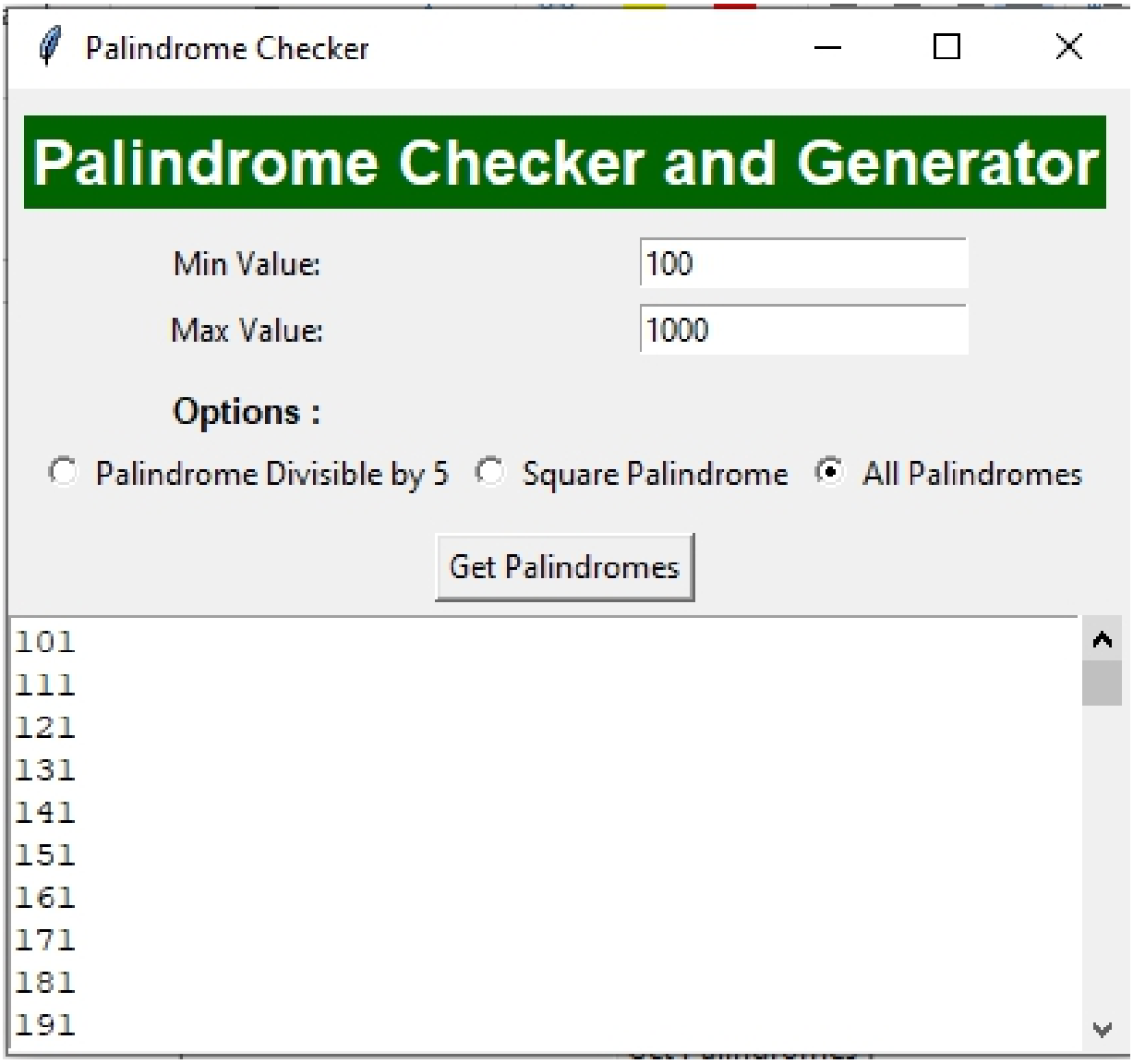


When the user selects the 'Square Palindrome' option and enters the range from 100 to 1000, clicking the 'Generate Palindrome' button will display the square palindrome numbers 121, 484, and 676 in the text area, as illustrated above. For each of these numbers, when we take the square root, we get a whole number that, when squared, equals the original number. This is the definition of a square palindrome. Lets try to solve this to understand the output

Example : 121 is a palindrome number. Its square root is 11, and when we square 11, we get 121, which is a whole number and equal to the original number.

484 is palimdrome number, square root of 484 is 22 and when we square 22, we get 484, which is whole number and equal to original number.

676 is a palindrome number, its square root is 26 and when we square 26, we get 676, which is whole number and equal to the original number. Therefore, 121, 484, and 676 are square palindromes.



After defining the range of numbers from 100 to 1000 and selecting the 'All Palindromes' option, the text area displays all palindrome

numbers such as 101, 111, 121, 131,141 etc., upon clicking the 'Get Palindrome' button, as depicted above.

Vowel Counter and Remover

Project #10

Design a simple Vowel Counter and Remover using Python tkinter

Introduction:

Welcome to the Vowel Counter and Remover project! This tool is designed to assist you in analyzing and manipulating text by counting vowels and removing them from the input.

In this project, users are invited to input any text of their choice. Upon entering the text, they have two options:

Count Vowels: By clicking on the "Count Vowels" button, users can instantly obtain a count of all vowels (i.e., 'a', 'e', 'i', 'o', 'u') present in the provided text.

Remove Vowels: Users can opt to remove all vowels from the input text by clicking on the "Remove Vowels" button.

Upon selecting their preferred action, users can observe the result directly on the main window. The count of vowels or the modified text without vowels will be displayed in a designated label area.

This project makes working with text easier! It is a handy tool for vowel analysis and manipulation in text processing. Let's build a powerful tool together!

Code:

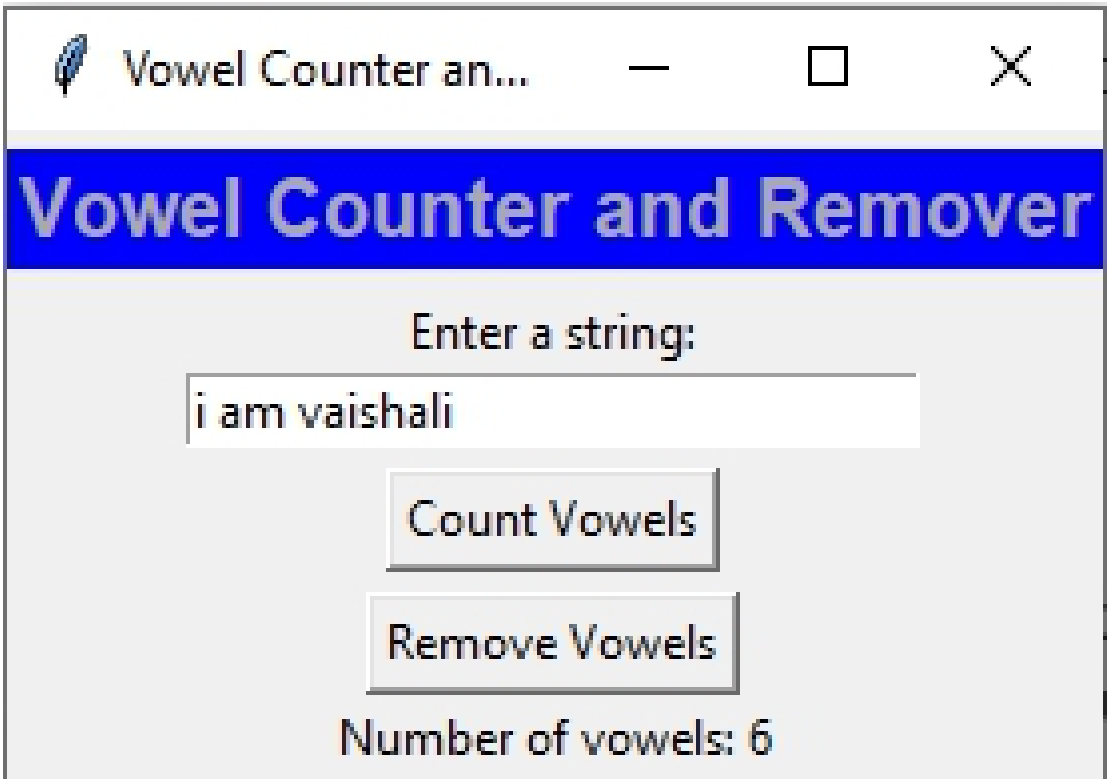
```
import tkinter as tk
# Define a string containing all vowels
vowels = "aeiouAEIOU"
# Function to count vowels
```

```
def count_vowels():
    # Get the input string from the entry widget
    input_string = entry.get()
    # Initialize a counter to keep track of the number of vowels
    num_vowels = 0
    # Loop through each character in the input string
    for char in input_string:
    # Check if the character is a vowel
        if char in vowels:
         # If it is, increment the vowel counter
            num_vowels += 1
    # Update the text of the result label
    result_label.config(text= f"Number of vowels: {num_vowels}")
# Function to remove vowels
def remove_vowels():
    input_string = entry.get()
    # Initialize an empty string to store the output without vowels
    output_string = ''
    # Loop through each character in the input string again
    for char in input_string:
    # Check if the character is not a vowel
        if char not in vowels:
        # If it's not a vowel, add it to the output string
            output_string += char
    result_label.config(text=f"String without vowels: {output_string}")
# Create main window
root = tk.Tk()
```

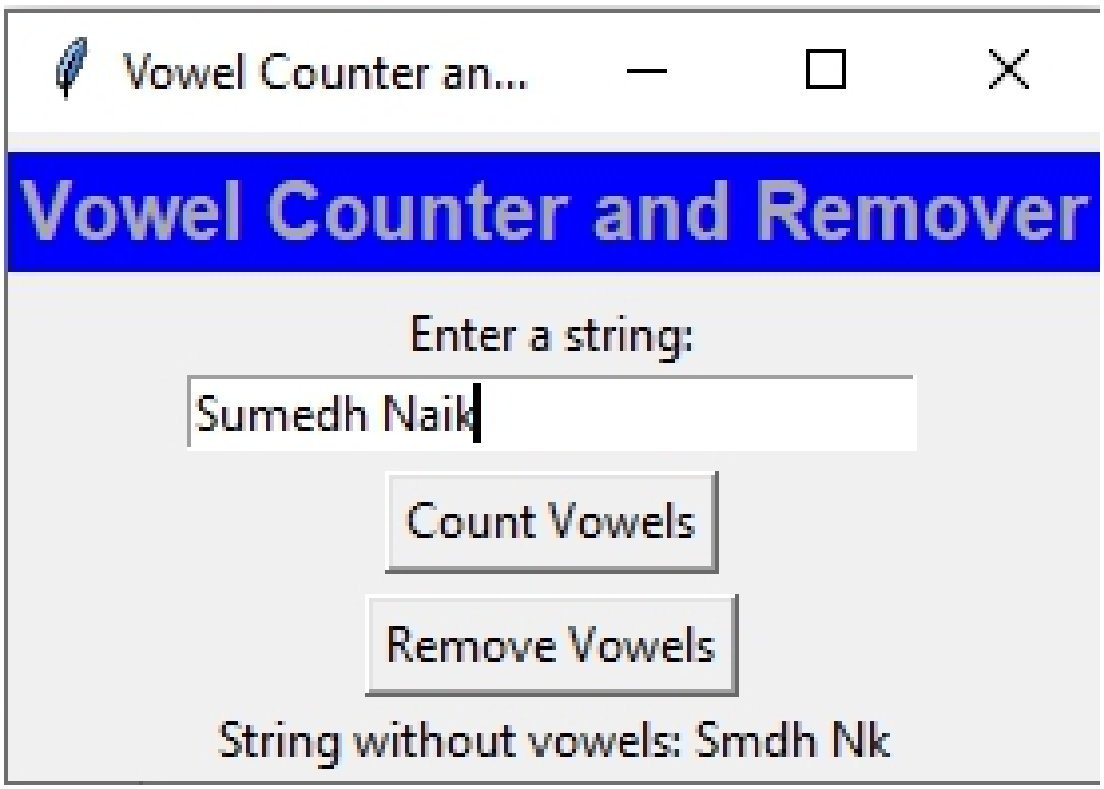
```
root.title("Vowel Counter and Remover")
# Create title for main window
title_label = tk.Label(root, text="Vowel Counter and
Remover")
title_label.pack(pady=5)
title_label.config(fg="Dark Gray", bg="blue", font=
("Arial",15,"bold"))
# Create input entry
entry_label = tk.Label(root, text="Enter a string:")
entry_label.pack()
entry = tk.Entry(root, width=30)
entry.pack()
# Create button to count vowels
button_count = tk.Button(root, text="Count Vowels",
command=count_vowels)
button_count.pack(pady=5)
# Create button to remove vowels
button_remove = tk.Button(root, text="Remove Vowels",
command=remove_vowels)
button_remove.pack()
# Create label to display result
result_label = tk.Label(root, text="")
result_label.pack()
root.mainloop()
```

Output:

After running the code, a window will pop up showing a simple interface. You'll see a box where you can type in any text you want to work with. Below that, there are two buttons: one for counting vowels and one for removing them. After clicking on either button, the result will appear in the same window. If you clicked "Vowel Count," you'll see the number of vowels displayed. If you clicked "Remove Vowel," you'll see the text with all the vowels removed.



In the window above, I typed the string "I am Vaishali" and then clicked on the "Count Vowels" button. The label below the button displayed the result: "Number of vowels: 6". We know that 'a', 'e', 'i', 'o', and 'u' are vowels, and in the given string, there are six vowels.



Likewise, in the window above, I input the new string "Sumedh Naik" and clicked on the "Vowel Remover" button. The program then removed all vowels from the string. Within this string, the vowels 'u', 'e', 'a', and 'i' were identified and subsequently removed. The resulting string, without any vowels, was displayed in the label below the button.

Binary to Decimal, Octal and Hexadecimal Convertor

Project #11

Create a graphical user interface (GUI) application using Tkinter in Python for converting a binary number to its decimal , octal,hexadecimal and text representations.

Introduction:

Welcome to our Binary to Decimal, Octal, and Hexadecimal Converter! In this Tkinter application, we'll design a simple tool to help you convert a binary number into its decimal, octal,hexadecimal and text representations. Let’s talk about Binary number system.

Binary numbers are fundamental in computer science and digital electronics, representing data using only two symbols: 0 and 1. Each digit in a binary number, known as a bit, holds a place value based on powers of 2.

For example, the binary number 1011 represents (1 × 2^3) + (0 × 2^2) + (1 × 2^1) + (1 × 2^0), which equals 11 in decimal notation.

Binary numbers are not intuitive for humans to read and understand. Converting them to decimal, octal, or hexadecimal makes them easier to understand and work with, especially in programming and digital design.

With Tkinter, we'll create an interface where you can enter any binary number. Then, with a click of the "Convert" button, you'll see the calculated binary, octal, and hexadecimal representations displayed in labels below,alongwith detail breakdown of decimal calculation steps. This breakdown will help you understand how the decimal number is formed from the binary input. This program will be handy for anyone who needs to perform these conversions quickly and easily.

Let's start converting binary numbers and exploring their different representations together.

Lets do some mathematics :

In Mathematics, a number system is used for expressing numbers. It is a way to represent numbers. We are not going into deep. The four different types of **number systems** are:

- Binary Number System (Base-2) : Uses two digits 0 and 1.
- Octal Number System (Base-8) : Utilize 8 digits ,o through 7.
- Decimal Number System (Base-10): Uses 10 digits 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9 .
- Hexadecimal Number System (Base-16): Utilizes sixteen digits, 0 through 9 and A through F.

Binary to Hexadecimal conversion

To convert binary to hexadecimal numbers, we need to use both the base numbers i.e 2 for binary and 16 for hexadecimal. The conversion process happens in two methods, the first method is by using the binary to hexadecimal conversion table where 1 hexadecimal number is equivalent to 4 binary numbers. The second method is by converting the binary number to a **decimal** number then convert it to a binary. Here We will use second method to do conversion.

Steps to Convert Binary to Hexadecimal without conversion table

Example :

$1011 = 1x2^3+0x2^2+1x2^1+1x2^0= 8+0+2+1=11$

$(1011)_2 = (11)_{10}$

Once the decimal number is obtained, we convert this decimal number to a hexadecimal number. The number is divided by 16 until the quotient is zero.

11/16 = 0 is the quotient, the remainder is 11

The final number is obtained by arranging the numbers from bottom to top i.e. 11. Since the hexadecimal number system only deals with 0 - 9 in numbers and 10 -15 in alphabets as A - F, so the number is B.

Binary to Octal

In the **binary number system**, we use only 0 and 1 digits to write numbers, while in the **octal number system** we use 0 to 7 digits to write numbers. We will see how to convert how to convert binary to octal numbers without conversion table.

Steps to Convert Binary to Octal without conversion table

In this, to convert binary to hexadecimal numbers, we need to convert binary to decimal as we have done in previous section then convert decimal to octal.

Example :

$1011 = 1x2^3+0x2^2+1x2^1+1x2^0= 8+0+2+1=11$

$(1011)_2 = (11)_{10}$

Once the decimal number is obtained, we convert this decimal number to a octal number. The number is divided by 8 until the quotient is 0.

11/8= 1 is the quotient, the remainder is 3

1/8 = 0 is the quotient, the remainder is 1

Finally,collect the remainder in reverse order ie.13

Therefore, the binary number$(1011)_2 = (13)_8$

Binary to Text conversion

Text is made up of characters, like letters and symbols. Each character has its own special number called an ASCII value. When software works with text, it turns each character into a binary number. This is because computers and devices only understand binary, so text needs to be converted to binary for them to process it.

To convert binary to text conversion, we need to convert binary to its decimal equivalent and then convert decimal to text.

Example :

$1011 = 1x2^3+0x2^2+1x2^1+1x2^0= 8+0+2+1=11$

$(1011)_2 = (11)_{10}$

After obtaining the decimal number, we can utilize the ASCII table to convert it into its corresponding character.

$(11)_{10} =$ Vertical Space control character

In this program, we will use built in function int(), hex(),oct() and char() to convert binary to their hexadecimal, and octal and text representations, respectively.
Here's a brief overview of each function:
Int(): This function takes a binary number and base 2 as input and return its decimal representation
Example :

```
>>> int("1011",2)

11
```

hex(): This function takes an integer(here decimal number) as input and returns its hexadecimal representation as a string prefixed with '0x'.

>>> hex(11)
'0xb’
oct(): This function takes an integer as input and returns its octal representation as a string prefixed with '0o'.
>>> oct(11)

'0o13'

chr() : This function takes an decimal number as input and convert ASCII value to its corresponding character.

>>> chr(65)

'A'

In this example, chr(65) returns the character 'A', as 65 is the ASCII value for the uppercase letter 'A'.

Solution:

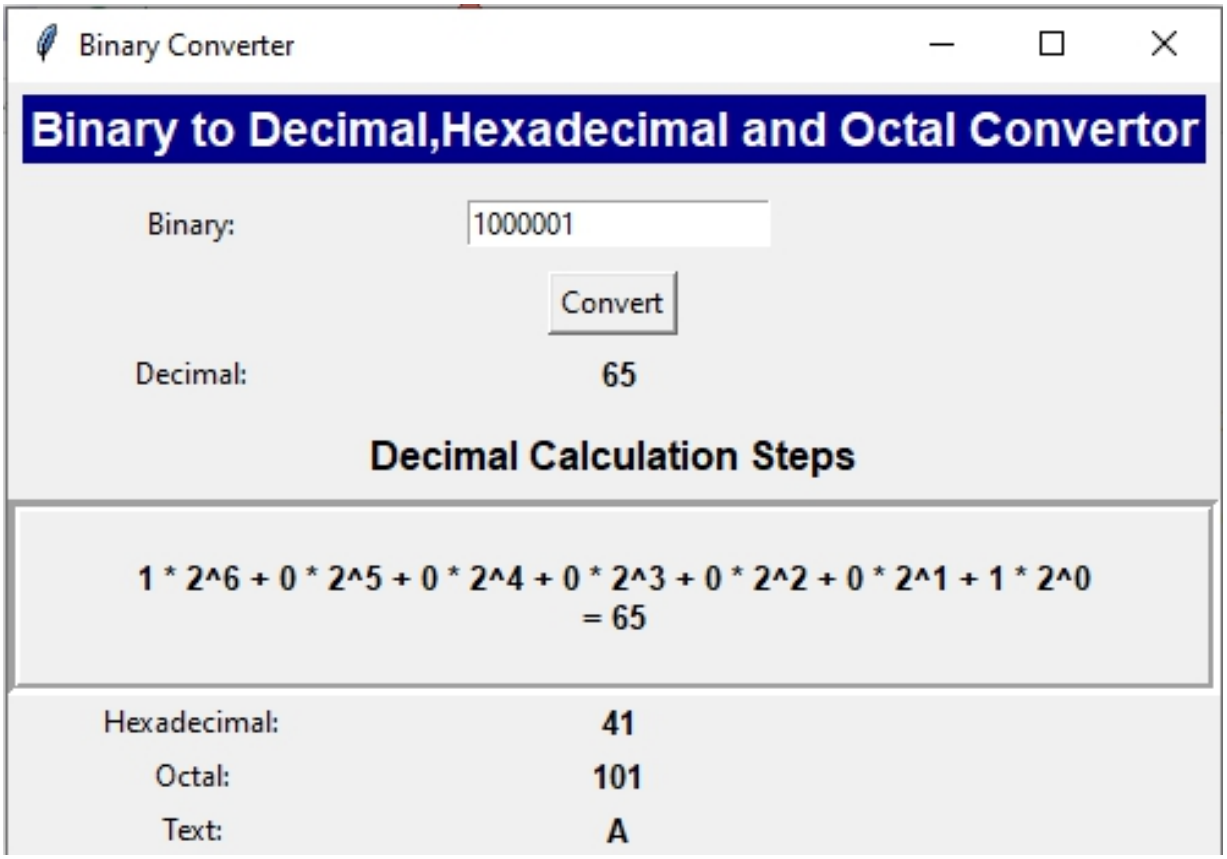
```
import tkinter as tk
def convert():
    # Get the binary input from the entry widget
    binary_input = binary_entry.get()
    try:
        # Calculate decimal value and display
        decimal_value = int(binary_input, 2)
        # Display calculated decimal value
        decimal_output.set(str(decimal_value))
        # Generate calculation steps
        # Count number of bit in binary input
        binary_length = len(binary_input)
        # Intitialize empty list to store individual calculation steps
        calculation_steps = []
        # Here highest power is 2 but indexing start form 0
        # so highest power is one less than number of bit
        power = binary_length - 1
```

```
        # Generate calculation steps for each bit in the binary input
        # This code append the calculation steps list
        # Show each step of calculation,
        # it show the current bit multiplied by 2 raised to the power of 'power'.
        for bit in binary_input:
            calculation_steps.append(f"{bit} * 2^{power}")
            # After each iteration of the loop,
            # the variable power is decremented by 1
            power -= 1

        # Combine calculation steps into a string
        calculation_steps_str = " + ".join(calculation_steps)
        calculation_steps_str += f" = {decimal_value}"

        # Set the calculation steps string to be displayed
        decimal_steps.set(calculation_steps_str)

        # Convert to other bases and display
        # Convert to hexdecimal and remove the first two character 'Ox' prefix
        hexadecimal_output.set(hex(decimal_value)[2:])
        # Convert to octal and remove first two character('Oo' prefix)
        octal_output.set(oct(decimal_value)[2:])
        # Convert to Ascii Character
        text_output.set(chr(decimal_value))
    except ValueError:
        # Handle invalid input
        decimal_output.set("Invalid Input")
        hexadecimal_output.set("Invalid Input")
        octal_output.set("Invalid Input")
        text_output.set("Invalid Input")
        decimal_steps.set("")
```

```
# Create main window
root = tk.Tk()
root.title("Binary Converter")
# Create title label for window
title_label = tk.Label(root, text="Binary to Decimal,Hexadecimal and
Octal Convertor")
title_label.grid(row=0, column=0, columnspan=4, pady=5)
title_label.config(fg="white", bg="Dark blue", font=("Helvetica", 12,
"bold"))
# Create label for text entry widget
binary_label = tk.Label(root, text="Binary:")
binary_label.grid(row=1, column=0)
binary_entry = tk.Entry(root)
binary_entry.grid(row=1, column=1, pady=10)
# Create Convert button
convert_button = tk.Button(root, text="Convert", command=convert)
convert_button.grid(row=2, column=0, columnspan=4)
# Create StringVar objects
decimal_output = tk.StringVar()
hexadecimal_output = tk.StringVar()
octal_output = tk.StringVar()
text_output = tk.StringVar()
decimal_steps = tk.StringVar()
# Create label for Decimal number
decimal_label = tk.Label(root, text="Decimal:")
decimal_label.grid(row=3, column=0)
# Create label for result
decimal_result = tk.Label(root, textvariable=decimal_output)
decimal_result.grid(row=3, column=1,pady=5)
decimal_steps_label = tk.Label(root, text="Decimal Calculation
Steps")
decimal_steps_label.grid(row=4, column=0,columnspan=4,pady=5)
# Text widget for displaying calculation steps
# Create a Frame to hold the output label with a border
```

```
steps_frame = tk.Frame(root, borderwidth=5, relief="groove")
steps_frame.grid(row=5, column=0, columnspan=4)
decimal_steps_text = tk.Label(steps_frame,
textvariable=decimal_steps,width=60, height=4,wraplength=400)
decimal_steps_text.grid(row=0, column=0)
hexadecimal_label = tk.Label(root, text="Hexadecimal:")
hexadecimal_label.grid(row=6, column=0)
hexadecimal_result = tk.Label(root,
textvariable=hexadecimal_output)
hexadecimal_result.grid(row=6, column=1)
octal_label = tk.Label(root, text="Octal:")
octal_label.grid(row=7, column=0)
octal_result = tk.Label(root, textvariable=octal_output)
octal_result.grid(row=7, column=1)
text_label = tk.Label(root, text="Text:")
text_label.grid(row=8, column=0)
text_result = tk.Label(root, textvariable=text_output)
text_result.grid(row=8, column=1)
root.mainloop()
```

Output: After running the above code, following GUI will be displayed on screen.



After initiating the conversion process by clicking the 'Convert' button, the main window dynamically presents the computed decimal alongwith calculation steps, octal, hexadecimal and text representations corresponding to the entered decimal number.

Let's try to understand the output :

If binary number is 1000001 then its decimal equivalent is 65 as shown in output(please check decimal calculation steps to understand)

Binary to Hexadecimal

We try to figure out how this hexadecimal number 41 is generated. We will use second method to do conversion(please refer introduction).

First, we convert this binary number to decimal number as shown below.

$1000001 = 1x2^6 + 0x2^5 + 0x2^4 + 0x2^3 + 0x2^2 + 0x2^1 + 1x2^0 = 64+0+0+1=65$

$(1000001)_2 = (65)_{10}$

Once the decimal number is obtained, we convert this decimal number to a hexadecimal number. The number is divided by 16 until the quotient is zero.

65/16 = 4 is the quotient, the remainder is 1

4/16 = 0 is the quotient, the remainder is 4

The final number is obtained by arranging the numbers from bottom to top i.e. 41.

Binary to Octal

Let's see ,how octal number 101 is generated

We repeat the same steps as before to convert binary to a decimal number. We already obtained 65.

Once the decimal number is obtained, we convert this decimal number to a octal number. The number is divided by 8 until the quotient is 0.

65/8= 8 is the quotient, the remainder is 1

8/8 = 1 is the quotient, the remainder is 0

1/8 = 0 is the quotient, the remainder is 1

Finally,collect the remainder in reverse order ie.101

Therefore, the binary number$(1000001)_2 = (101)_8$

Binary to Text conversion:

In the output, we received 65 as a decimal number. Since we know how to convert binary to decimal, we then checked the ASCII equivalent of this decimal number. Recognizing that the ASCII equivalent of capital A is 65, we conclude that the answer is A.

Volume and Surface Area Calculator for 3D Shapes

Project #12

Design Volume and Surface Area Calculator using Python and Tkinter

Introduction:

Welcome to the Volume and Surface Area Calculator!

This calculator is designed to help you effortlessly compute the volume and surface area of various geometric shapes, including cubes, spheres, cuboids, cones, and cylinders.

Simply select the desired shape from the dropdown menu, and the corresponding input fields will dynamically adjust to accommodate the necessary parameters. For instance, when you choose "Cube," only the side length entry field will be enabled, while the others will be disabled to streamline your input process.

Once you've entered the required values, just click the "Calculate" button, and voila! The calculator will do the mathematical calculation and display the volume and surface area of the chosen shape right below the button, making it easy for you to obtain accurate results in no time.

Whether you're studying geometry or working on complex shapes, this calculator makes it easy to get the answers you need. Let's start exploring shapes and numbers together!

Before writing the code, it's essential to understand the concepts of volume and surface area and how to calculate them.

Volume refers to the amount of space occupied by a three-dimensional object, while **surface area** refers to the total area

covering the exterior of the object. To calculate them, you typically need to know the specific formulas corresponding to the shape of the object, such as a cube, sphere, cylinder, etc. Once you grasp these concepts and formulas, you can proceed to implement them in your code.

Volume and Surface Area Formulas for 3D Shapes :

Cube :

Volume= side x side x side

Surface Area = 6 x side x side

Example :

Given : side = 5cm

Volume of cube = 5cm x 5cm x 5cm = 125 cm^3

Surface Area = 6 x 5cm x 5cm = 150cm^2

Cylinder :

Volume = π x $radius^2$ x height

Surface Area = 2 π x radius x height + 2 π radius x radius

Example :

Given : radius = 5cm , height = 10cm

Volume = 3.14 x 5cm x 5cm x 10cm = 3.14 x 25 x 10 = 785cm^3

Surface Area

= 2 x 3.14 x 5cm x 10cm + 2 x 3.14 x 5cm x 5cm

= 3.14 x 100 + 3.14 x 50

= 314 + 157 = 471cm^2

Sphere :

Volume = 4/3 x π x radius^3

Surface Area = 4 π x radius x radius

Example:

Given : radius = 4.5cm

Volume = 4/3 x 3.14 x 4.5cm x 4.5cm x 4.5cm = 381.7cm^3

Surface Area = 4 x 3.14 x 4.5cm x 4.5cm = 254.7cm^2

Cuboid :

Volume = length x width x height

Surface Area

= 2 x (length x width + width x height + height x length)

Example:

Given : length = 3cm , width = 3cm, height = 6cm

Volume = 3cm x 3cm x 6cm = 54cm^3

Surface Area

= 2((3 × 3) + (3 × 6) + (6 × 3))
= 2(9 + 18 + 18)
= 2 x 45 = 90 cm^3

Cone :

Volume = 1/3 x π x radius^2 x height

Surface Area = $\boldsymbol{\pi}$ x radius x (radius + (radius^2 + height^2)**0.5)

Example:

Given : radius = 2cm , height = 5cm

Volume

=1/3 x 3.14 x 2cm x 2cm x 5cm =

=1/3 x 3.14 x 20cm^3

= 20.94cm^3

Surface Area

= 3.14 x 2cm x (2cm +(16cm^2 + 25cm^2) **0.5)

= 3.14 x 2cm x (2cm + (41cm^2)**0.5)= 46.40cm^2

Requirement :

Module used: math

The math module in Python provides various mathematical functions and constants for performing mathematical operations. It includes functions for trigonometry, logarithms, exponentials, and more. Additionally, it provides constants like π (pi) and e.

The math module is part of Python's standard library, so it's available by default when you install Python. You can import and use it in your Python scripts without needing to install it separately.

You can simply import it like this:

import math

Or, if you only need certain functions or constants from the math module, you can import them individually:

from math import pi

Code:

```
import tkinter as tk
from math import pi
def shape_changed(*args):
    shape = shape_var.get()  # Get the selected shape
    if shape == "Cube":  # Check if the selected shape is "Cube"
        # Enable side entry and disable other entries
        side_entry.config(state="normal")
        radius_entry.config(state="disabled")
        height_entry.config(state="disabled")
        length_entry.config(state="disabled")
        width_entry.config(state="disabled")
    elif shape == "Sphere":# Check if the selected shape is "Sphere"
        # Enable radius entry and disable other entries
        side_entry.config(state="disabled")
        radius_entry.config(state="normal")
        height_entry.config(state="disabled")
        length_entry.config(state="disabled")
        width_entry.config(state="disabled")
```

```
    elif shape == "Cylinder":# Check if the selected shape is "Cyliinder"
        # Enable radius and height entry and disable other entries
        side_entry.config(state="disabled")
        radius_entry.config(state="normal")
        height_entry.config(state="normal")
        length_entry.config(state="disabled")
        width_entry.config(state="disabled")
    elif shape == "Cuboid":# Check if the selected shape is "Cuboid"
        # Enable height,length and width entry and disable other entries
        side_entry.config(state="disabled")
        radius_entry.config(state="disabled")
        height_entry.config(state="normal")
        length_entry.config(state="normal")
        width_entry.config(state="normal")
    elif shape == "Cone":# Check if the selected shape is "Cone"
        # Enable radius and height entry and disable other entries
        side_entry.config(state="disabled")
        radius_entry.config(state="normal")
        height_entry.config(state="normal")
```

```
        length_entry.config(state="disabled")
        width_entry.config(state="disabled")
# Calculate function
def calculate():
    shape = shape_var.get()
    if shape == "Cube":
        side = float(side_entry.get())
        volume = side ** 3
        surface_area = 6 * side ** 2
    elif shape == "Sphere":
        radius = float(radius_entry.get())
        volume = (4/3) * pi * radius ** 3
        surface_area = 4 * pi * radius ** 2
    elif shape == "Cylinder":
        radius = float(radius_entry.get())
        height = float(height_entry.get())
        volume = pi * radius ** 2 * height
        surface_area = 2 * pi * radius * (radius + height)
    elif shape == "Cuboid":
        length = float(length_entry.get())
        width = float(width_entry.get())
```

```
        height = float(height_entry.get())
        volume = length * width * height
        surface_area = 2 * (length * width + width * height + height * length)
    elif shape == "Cone":
        radius = float(radius_entry.get())
        height = float(height_entry.get())
        volume = (1/3) * pi * radius ** 2 * height
        surface_area = pi * radius * (radius + (radius ** 2 + height ** 2) ** 0.5)
    volume_label.config(font=("TkDefaultFont", 12, "bold"),text="Volume: {:.2f}".format(volume))
    surface_area_label.config(font=("TkDefaultFont", 12, "bold"),text="Surface Area: {:.2f}".format(surface_area))
    # Clear entry fields after calculating
    side_entry.delete(0, tk.END)
    radius_entry.delete(0, tk.END)
    height_entry.delete(0, tk.END)
    length_entry.delete(0, tk.END)
    width_entry.delete(0, tk.END)
# Create the main window
root = tk.Tk()
root.title("Shape Calculator")
```

 # Create title label

```
title_label = tk.Label(root, text="Volume and Area of Geometric Shape Calculator")

title_label.grid(row=0, column=0, columnspan=2, pady=10)

title_label.config(fg="white", bg="Dark red", font=("Arial",16, "bold"))
```

 # Create widgets

 # Create Shape label and dropdown menu

```
shape_label = tk.Label(root, text="Select shape:")

shape_label.grid(row=1, column=0, padx=10, pady=5, sticky="w")

shape_label.config(font=("TkDefaultFont", 10, "bold"))

shape_var = tk.StringVar(value="Cube")

shape_options = ["Cube", "Sphere", "Cylinder", "Cuboid", "Cone"]

shape_menu = tk.OptionMenu(root, shape_var, *shape_options, command=shape_changed)

shape_menu.grid(row=1, column=1, padx=10, pady=5, sticky="w")

shape_menu.config(font=("TkDefaultFont", 10, "bold"))
```

 # Create side label and entry widget for cube

```
side_label = tk.Label(root, text="Side (Cube):")

side_label.grid(row=2, column=0, padx=10, pady=5, sticky="w")

side_label.config(font=("TkDefaultFont", 10, "bold"))

side_entry = tk.Entry(root)

side_entry.grid(row=2, column=1, padx=10, pady=5, sticky="w")
```

 # Create radius label and entry widget

radius_label = tk.Label(root, text="Radius:")

radius_label.grid(row=3, column=0, padx=10, pady=5, sticky="w")

radius_label.config(font=("TkDefaultFont", 10, "bold"))

radius_entry = tk.Entry(root)

radius_entry.grid(row=3, column=1, padx=10, pady=5, sticky="w")

 # Create height label and entry widget

height_label = tk.Label(root, text="Height:")

height_label.grid(row=4, column=0, padx=10, pady=5, sticky="w")

height_label.config(font=("TkDefaultFont", 10, "bold"))

height_entry = tk.Entry(root)

height_entry.grid(row=4, column=1, padx=10, pady=5, sticky="w")

 # Create length label and entry widget

length_label = tk.Label(root, text="Length (Cuboid):")

length_label.grid(row=5, column=0, padx=10, pady=5, sticky="w")

length_label.config(font=("TkDefaultFont", 10, "bold"))

length_entry = tk.Entry(root)

length_entry.grid(row=5, column=1, padx=10, pady=5, sticky="w")

 # Create width label and entry widget

width_label = tk.Label(root, text="Width (Cuboid):")

width_label.grid(row=6, column=0, padx=10, pady=5, sticky="w")

```
width_label.config(font=("TkDefaultFont", 10, "bold"))

width_entry = tk.Entry(root)

width_entry.grid(row=6, column=1, padx=10, pady=5, sticky="w")

bold_font = ("Helvetica", 12, "bold") # Adjust the font family, size, and weight as needed

# Create calculate button

calculate_button = tk.Button(root, text="Calculate",
command=calculate, width =10, height=2, font=bold_font)

calculate_button.grid(row=7, columnspan=2, padx=10, pady=10)

# Create Volume and surface area label

volume_label = tk.Label(root, text="Volume:")

volume_label.grid(row=8, columnspan=2, padx=10, pady=5)

surface_area_label = tk.Label(root, text="Surface Area:")

surface_area_label.grid(row=9, columnspan=2, padx=10, pady=5)

# Disable initial entry fields based on default shape

shape_changed()

root.mainloop()
```

Output :

After running the provided code for the above project, you will see a graphical user interface (GUI) with the following components:

Combobox : At the top of the GUI, you'll find a combobox. This handy dropdown menu lets you choose the shape you want to work with. Simply click on the dropdown arrow, and you'll see a list of available shapes to choose from: cube, cuboid, cone, cylinder, and

sphere. Once you've made your selection, the corresponding entry fields will appear below, allowing you to enter the dimensions of your chosen shape.

Labels and Entry widgets: Below the combobox, you'll see a row of entry widgets and corresponding labels. These labels are labeled "Side," "Height," "Radius," "Width," and "Length," respectively. The number of entry widgets enabled depends on the shape chosen from the combobox. For instance:

If "Cube" is selected, only one entry widget for entering the side length will be enabled.

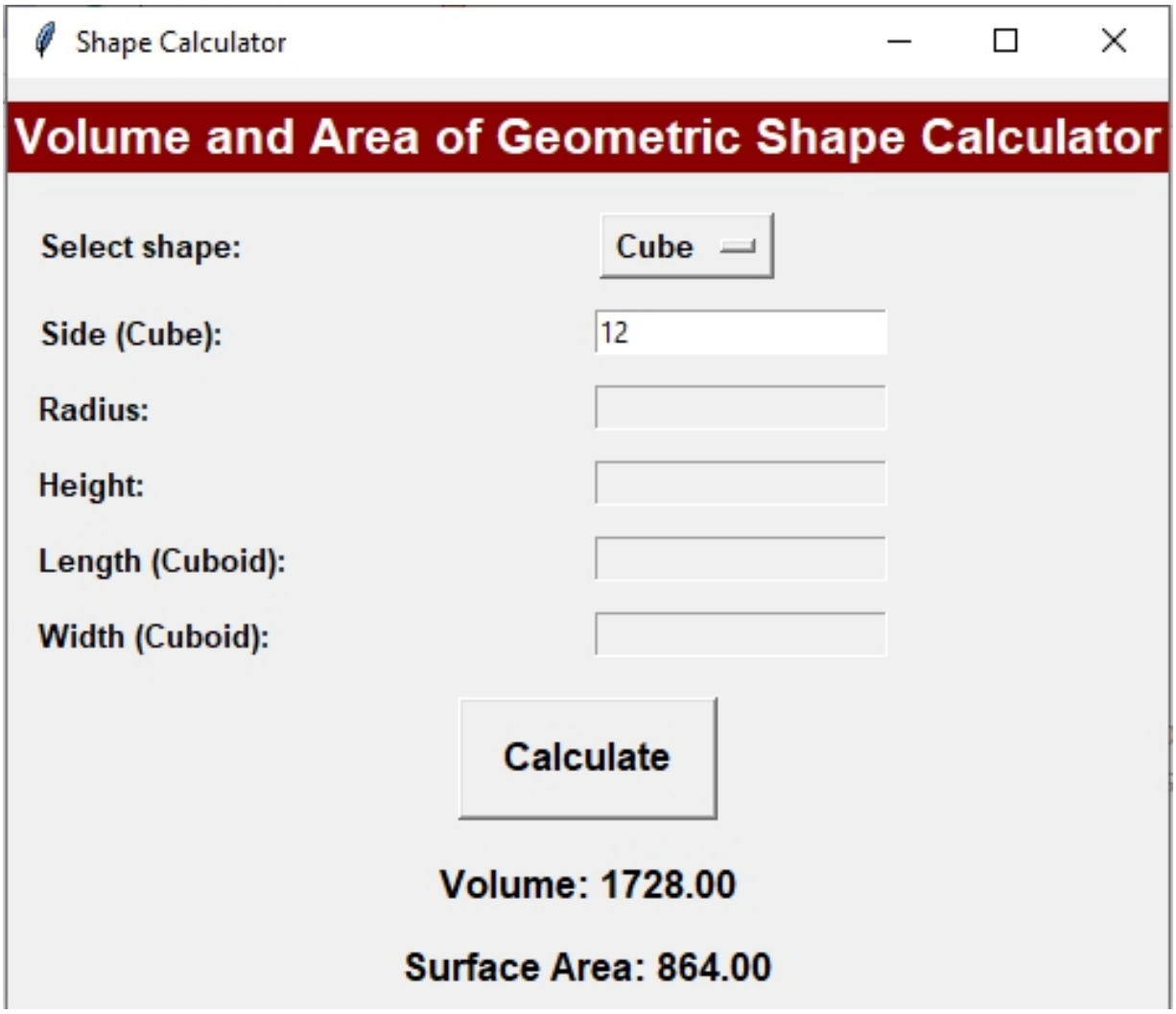
If "Cuboid" is selected, entry widgets for entering the height, length, and width will be enabled.

For other shapes, the relevant entry widgets will be

enabled based on their specific dimensions.

Calculate Button : Beneath the entry widget, you'll find the "Calculate" button. Simply click this button to trigger the calculation of both the volume and surface area based on the dimensions you've entered.

Result Label: Directly beneath the "Calculate" button, you'll see a large result label. Once you click the button, this label will display the calculated results in a clear and easy-to-read format.

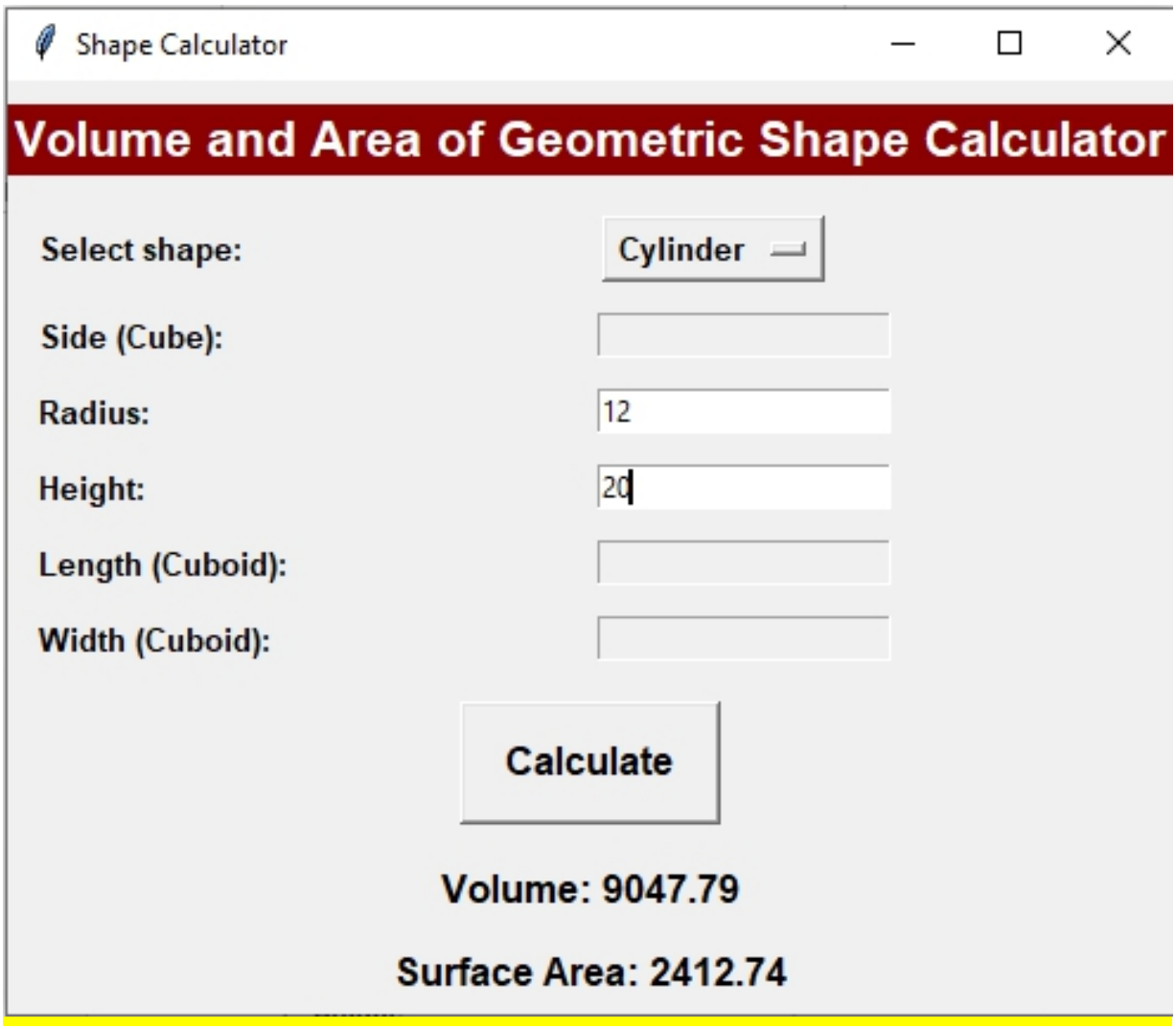


Let's manually calculate the output for the cube. The user has entered a side length value of 12. We'll use formulas to calculate the volume and surface area to confirm if we get the correct output or not.

Given : side = 12cm

Volume of cube = 12cm x 12cm x 12cm = 1728 cm^3

Surface Area = 6 x 12cm x 12cm = 864cm^2



Example :

Given : radius = 12cm , height = 20cm

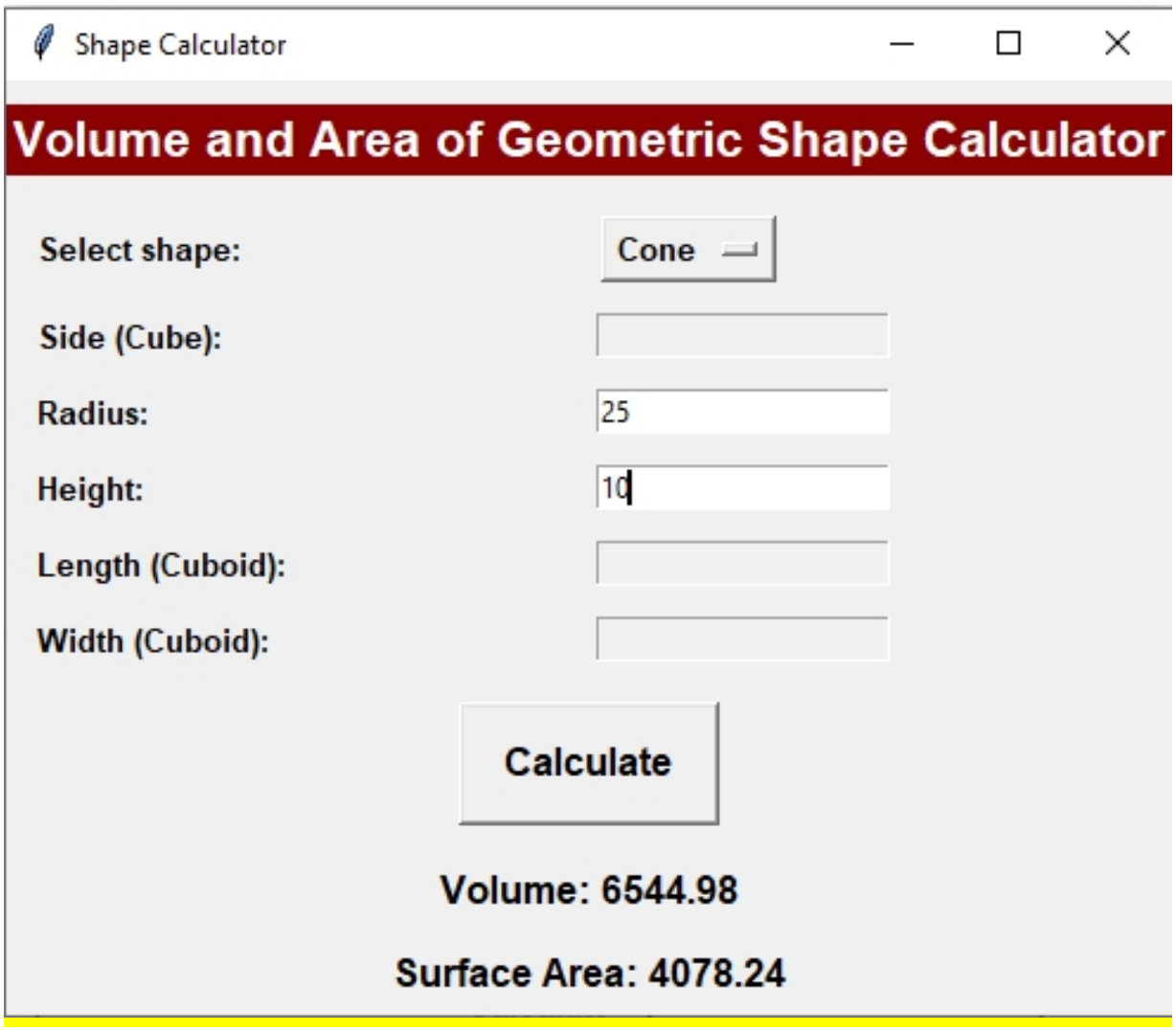
Volume = 3.14 x 12cm x 12cm x 20cm = 9047.79 cm^3

Surface Area of Cylinder

= 2 x 3.14 x 12cm x 20cm + 2 x 3.14 x 12cm x 12cm

= 3.14 x 480 + 3.14 x 288

= 1507.2 + 904.32 = 2412.74 cm^2



Example:

Given : radius = 25cm , height = 10cm

Volume of Cone

=1/3 x 3.14 x 25cm x 25cm x 10cm =

=1/3 x 3.14 x 6250cm^3

= 6544.98 cm^3

Surface Area of Cone

= 3.14 x 25cm x (25cm +($625cm^2$ + $100cm^2$) **0.5)

= 3.14 x 25cm x (25cm + ($725cm^2$)**0.5)= $4078.24cm^2$

Image Resizer

Project #13

Design a GUI application using Tkinter in Python that allows users to resize image dynamically.

Introduction:

Welcome to the Image Resizer! This handy tool helps you easily resize your images offline, and it's completely free to use.

In today's digital age, manipulating images to fit various dimensions is a common need, whether it's for social media, websites, or personal projects. With our user-friendly tool, you can resize images with ease, ensuring they meet your specific size requirements.

Using the Image Resizer is simple. Begin by selecting the image you wish to resize, then specify the desired width and height. Once you've entered your preferences, click the "Resize" button, and voila! Our tool will quickly adjust the image to your specified dimensions.

Not only does the Image Resizer resize your images, but it also provides you with the exact dimensions of the resized image, so you can ensure it meets your requirements perfectly. Let's get started with the Image Resizer!

Requirement:

PILLOW Library

In this program, we will use Tkinter pillow library for image processing. So let’s talk about Tkinter Pillow. **Pillow (formerly known as PIL)** is a popular library for image processing and manipulation in Python. It supports a wide range of image formats, including PNG, JPEG, GIF, and more.
You can use Pillow to open, resize, crop, convert, and perform other operations on images. Tkinter and Pillow complement each other

perfectly. You can use Pillow to process and manipulate images, then use Tkinter to display them in your GUI.
To install the Pillow library, you can use Python's package manager, pip. Here's how you can do it:

pip install pillow

This command will download and install the Pillow library and all its dependencies. Once the installation is complete, we can start using Pillow in our Python projects for image processing tasks.
When you want to work with images using the Pillow library (PIL stands for Python Imaging Library, which was the old name for Pillow), you import the Image module from PIL. Similarly, if you're working with Tkinter and want to display images within Tkinter GUI applications, you import the ImageTk module from PIL. Here's the standard import statement:

from PIL import Image, ImageTk

With this import statement, you can use Image class to load image from file, resize it and apply filter and do many more tasks and ImageTk class to convert a Pillow Image object into a format compatible with Tkinter's PhotoImage widget, which you can then display in your GUI. Both Image and ImageTk classes are imported from Pillow Library. We will discuss what PhotoImage widget is later.

Code :

```
import tkinter as tk

from tkinter import filedialog

from PIL import Image, ImageTk

def resize_image():

    try:

        # Get the selected image file path
```

```
        image_path = file_path.get()
        # Open the image file
        image = Image.open(image_path)
        # Get the desired height and width
        new_height = int(entry_height.get())
        new_width = int(entry_width.get())
        # Resize the image
        resized_image = image.resize((new_width, new_height), Image.LANCZOS)
        # Display the resized image
        resized_image.thumbnail((300, 300))
        resized_photo = ImageTk.PhotoImage(resized_image)
        label_resized.config(image=resized_photo)
        label_resized.image = resized_photo
        # Display actual height and width of the resized image
        resized_width, resized_height = resized_image.size
        label_resized_size.config(text=f" Resized Image Size: {resized_width}x{resized_height} pixels")
    except Exception as e:
        # Show error if any
        status_label.config(text="Error: " + str(e))
def browse_file():
```

```
    # Open a file dialog to select an image file
    file_path.set(filedialog.askopenfilename())
    # Load and display the selected image
    image_path = file_path.get()
    if image_path:
        image = Image.open(image_path)
        image.thumbnail((300, 300))
        photo = ImageTk.PhotoImage(image)
        label_selected_image.config(image=photo)
        label_selected_image.image = photo
        # Display actual height and width of the selected image
        image_width, image_height = image.size
        status_label.config(text=f" Original Image Size: {image_width}x{image_height} pixels")
# Create the main window
root = tk.Tk()
root.title("Image Resizer")
# Variables
file_path = tk.StringVar()
# Create Label for Project title
# Apply font style to title label
```

```
font_bold = ("Helvetica",18,"bold")
label_title = tk.Label(root, text="Image Resizer")
label_title.config(fg="white",bg="Navy Blue",font=font_bold)
label_title.grid(row=0, columnspan=2, padx=5, pady=5)
# Create Label for Project title
# Apply font style to title label
font_bold = ("Calibri",15,"bold")
label_title = tk.Label(root, text="Easily Resize Image Offline for free")
label_title.config(fg="Navy Blue",font=font_bold)
label_title.grid(row=1, columnspan=2, padx=5, pady=5)
# Widgets
# Create label and place it on window layout
label_select = tk.Label(root, text="Select Image:")
label_select.grid(row=2, column=0, padx=5, pady=5)
# Create browse button
btn_browse = tk.Button(root, text="Browse", command=browse_file)
btn_browse.grid(row=2, column=1, padx=5, pady=5)
# Create status label
status_label = tk.Label(root, text="", fg="green")
status_label.grid(row=4, columnspan=2, padx=5, pady=5)
# Create label to show selected image on form
```

```
label_selected_image = tk.Label(root)
label_selected_image.grid(row=2, columnspan=2, padx=5, pady=5)
# Create width label and its entry widget
label_width = tk.Label(root, text="Width (px):")
label_width.grid(row=5, column=0, padx=5, pady=5)
entry_width = tk.Entry(root)
entry_width.grid(row=5, column=1, padx=5, pady=5)
# Create height label and its entry widget
label_height = tk.Label(root, text="Height (px):")
label_height.grid(row=6, column=0, padx=5, pady=5)
entry_height = tk.Entry(root)
entry_height.grid(row=6, column=1, padx=5, pady=5)
# Create resize button
btn_resize = tk.Button(root, text="Resize", command=resize_image)
btn_resize.grid(row=7, columnspan=2, padx=5, pady=5)
# Create label to show resized image
label_resized_image = tk.Label(root, text="",fg="green")
label_resized_image.grid(row=6, columnspan=2, padx=5, pady=5)
label_resized = tk.Label(root)
label_resized.grid(row=9, columnspan=2, padx=5, pady=5)
# Create label to show resized image size
```

```
label_resized_size = tk.Label(root, text="", fg="green")

label_resized_size.grid(row=10, columnspan=2, padx=5, pady=5)

# Start the main event loop

root.mainloop()
```

Output:

After executing the provided code, the graphical user interface (GUI) of the Image Resizer application appears with the following layout and functionalities:

Application Title : At the very top of the window, the title "Easily Resize Image Offline for Free" is displayed prominently in a larger font.

Image Selection Section: Below the title, there is a horizontal section containing: A label that says "Select Image". A "Browse" button next to the label.

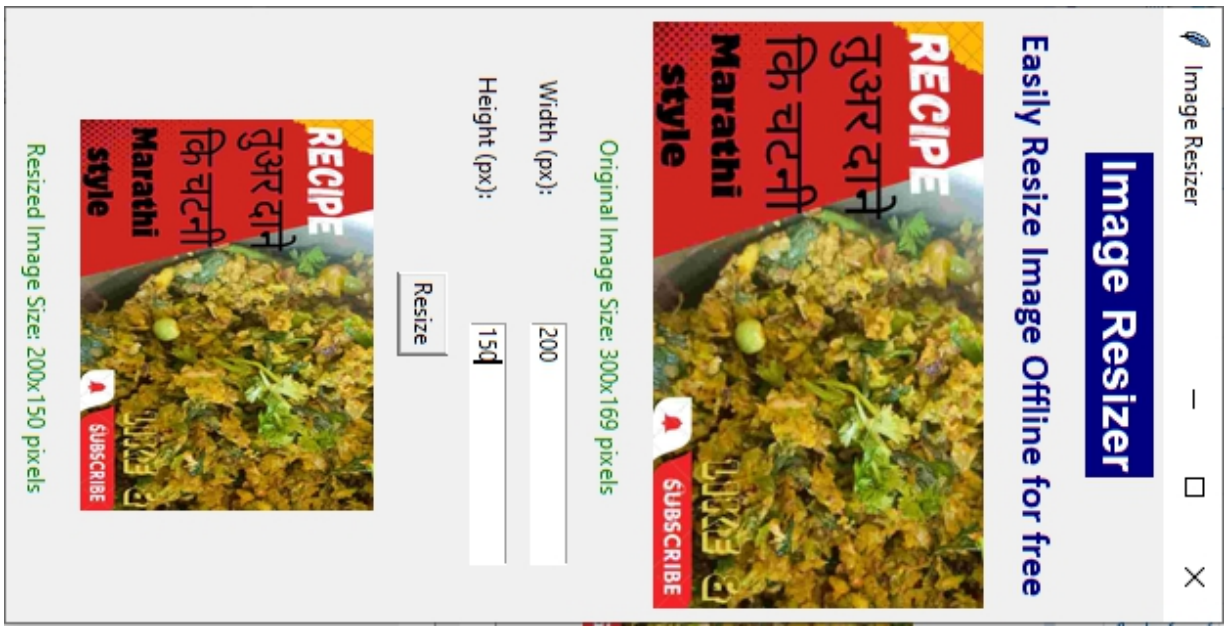
Image Display: When the user clicks the "Browse" button, a file dialog opens, allowing the user to select an image file from their computer. Once an image is selected, it is displayed in place of the "Select Image" label and the "Browse" button. The image is scaled down to fit within the allocated space without altering its aspect ratio.

Dimension Input Fields: Below the image display section, there are two input fields for the user to specify the new dimensions of the image: A label "Width (px)" followed by an entry field where the user can type the desired width in pixels. A label "Height (px)" followed by an entry field where the user can type the desired height in pixels.

Resize Button: Directly beneath the dimension input fields, there is a "Resize" button. When clicked, this button triggers the resizing operation based on the dimensions entered by the user.

Resized Image Display: Below the "Resize" button, there is a label that updates to "Resized Image" once the resizing operation is complete, showing the new dimensions.The resized image itself is displayed above this label.

This is main GUI window, where users can upload image by clicking on the browse button.



When the user selects an image from the file open dialog box, the image is uploaded as shown above. The original image size is also displayed to the user, allowing them to resize the image as needed. In the GUI above, the user entered dimensions of 200px width and 150px height to resize the image. After clicking the resize button, the resized image is displayed with the specified dimensions, and the label shows the new image size.

Pencil Sketch Art Maker

Project #14

Develop an Pencil Sketch Art Maker Application using Tkinter and Python

Introduction:

Welcome to Pencil Sketch Art Maker! It's a cool tool that turns your regular photos into awesome pencil drawings. With this app, you can make your pictures look like they were drawn by hand, even if you're not an artist! The goal of Pencil Sketch Art Maker is to make it super easy for anyone to create cool pencil sketches from their photos. Whether you're a pro artist or just someone who likes playing with pictures, this app is for you.

All you need to do is pick a photo from your computer, click a button, and voila! Pencil sketch maker does all the hard work for you, turning your photo into a beautiful pencil sketch in just a few seconds.

By the end of this project, you'll know how to use a simple tool to create pencil sketch from photo and images. Let's start building the creative app!

Requirement :

Modules used:

tkinter: In this project, tkinter is used to create the main application window, buttons for opening images and creating sketches, labels for displaying text, and a canvas for displaying images.

filedialog (from tkinter):

We use the filedialog.askopenfilename() function to open a file dialog window when the user clicks the "Open Image" button. This allows the user to select an image file from their computer.

messagebox (from tkinter):

We use the messagebox.showerror() function to display an error message if an error occurs during image processing. This informs the user about the issue and helps in troubleshooting.

PIL (Python Imaging Library): PIL is used to open and manipulate images in various ways. We use it to open images from file paths, resize images to fit the canvas, and convert images to Tkinter-compatible format for display. We use the ImageFilter module to make images blurry, sharp, or find edges. We also use the ImageOps module to change images to black and white, switch colors, or make them clearer.

All of these modules (Image, ImageTk, ImageFilter, and ImageOps) are part of the Python Imaging Library (PIL), now known as the Pillow library. When you install Pillow using pip (pip install Pillow), you get access to all these modules. You don't need to install them separately. They come bundled with the Pillow library, making it convenient to work with images in Python.

To use all of these modules in our code, we simply import them using the following statement:

from PIL import Image, ImageTk, ImageFilter, ImageOps

cv2 (OpenCV): OpenCV is a popular library for computer vision and image processing tasks. It is used for advanced image processing tasks, such as converting images to grayscale, applying filters, and creating the pencil sketch effect. We use OpenCV functions like cv2.cvtColor() for color conversions, cv2.GaussianBlur() for blurring, and cv2.divide() for blending images to create the sketch effect. We will discuss this module in detail in next project.

If you intend to use the OpenCV library (cv2), you will need to import it separately. OpenCV is not part of the Python Imaging Library (PIL) or Pillow. You can install OpenCV using pip

pip install opencv-python.

Then, you can import it in your code using import cv2.

numpy:

Numpy is a powerful library for numerical computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.

In this project, OpenCV uses numpy arrays to handle image data effectively. OpenCV functions work with these numpy arrays, both taking them as input and returning them as output, for different image processing tasks.

You can install NumPy using pip

pip install numpy

and then import it into your code using import numpy or import numpy as np if you prefer to use an alias.

Code:

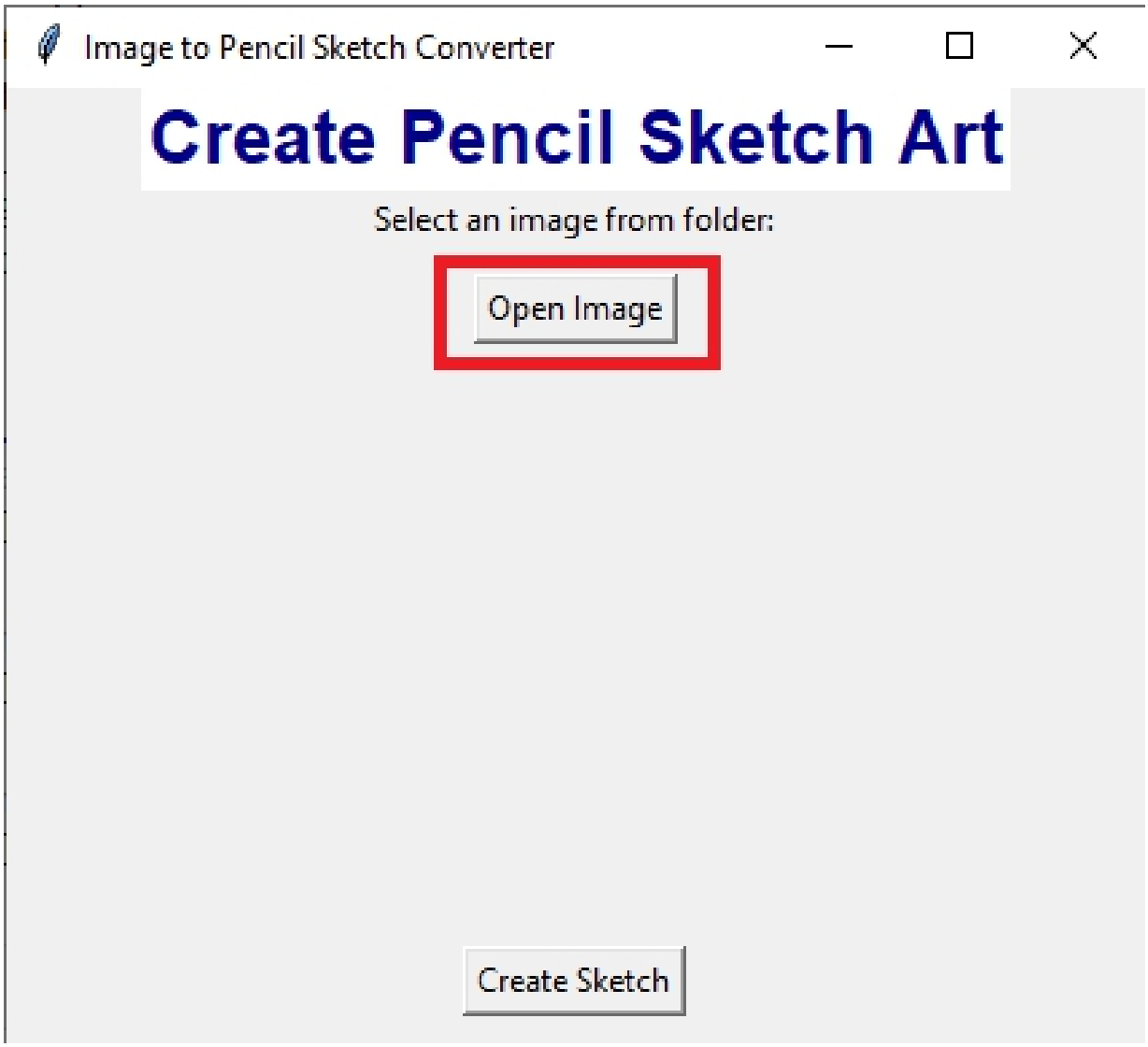
```
import tkinter as tk
from tkinter import filedialog, messagebox
from PIL import Image, ImageTk, ImageFilter, ImageOps
import cv2
import numpy as np

file_path = None  # Global variable to store the file path

def convert_to_sketch(original_img):
    try:
        # Convert the image to grayscale
        gray_img = cv2.cvtColor(original_img,
cv2.COLOR_BGR2GRAY)

        # Invert the grayscale image
```

```
        inverted_gray_img = 255 - gray_img

        # Blur the inverted grayscale image
        blurred_img = cv2.GaussianBlur(inverted_gray_img,
(21, 21), 0)

        # Invert the blurred image
        inverted_blurred_img = 255 - blurred_img

        # Blend the inverted blurred image with the
grayscale image using Color Dodge
        pencil_sketch_img = cv2.divide(gray_img,
inverted_blurred_img, scale=256.0)

        return pencil_sketch_img
    except Exception as e:
        messagebox.showerror("Error", f"An error occurred:
{str(e)}")
        return None

def open_file():
    global file_path
    file_path = filedialog.askopenfilename(filetypes=[("Image
Files", "*.jpg;*.jpeg;*.png;*.bmp;*.gif")])
    if file_path:
        original_img = cv2.imread(file_path)
        if original_img is not None:
            display_original_image(original_img)

def display_original_image(original_img):
    # Resize the image to fit in the canvas
    original_img = cv2.resize(original_img, (200, 200))

    # Convert image to PIL format
    original_img_pil =
Image.fromarray(cv2.cvtColor(original_img,
```

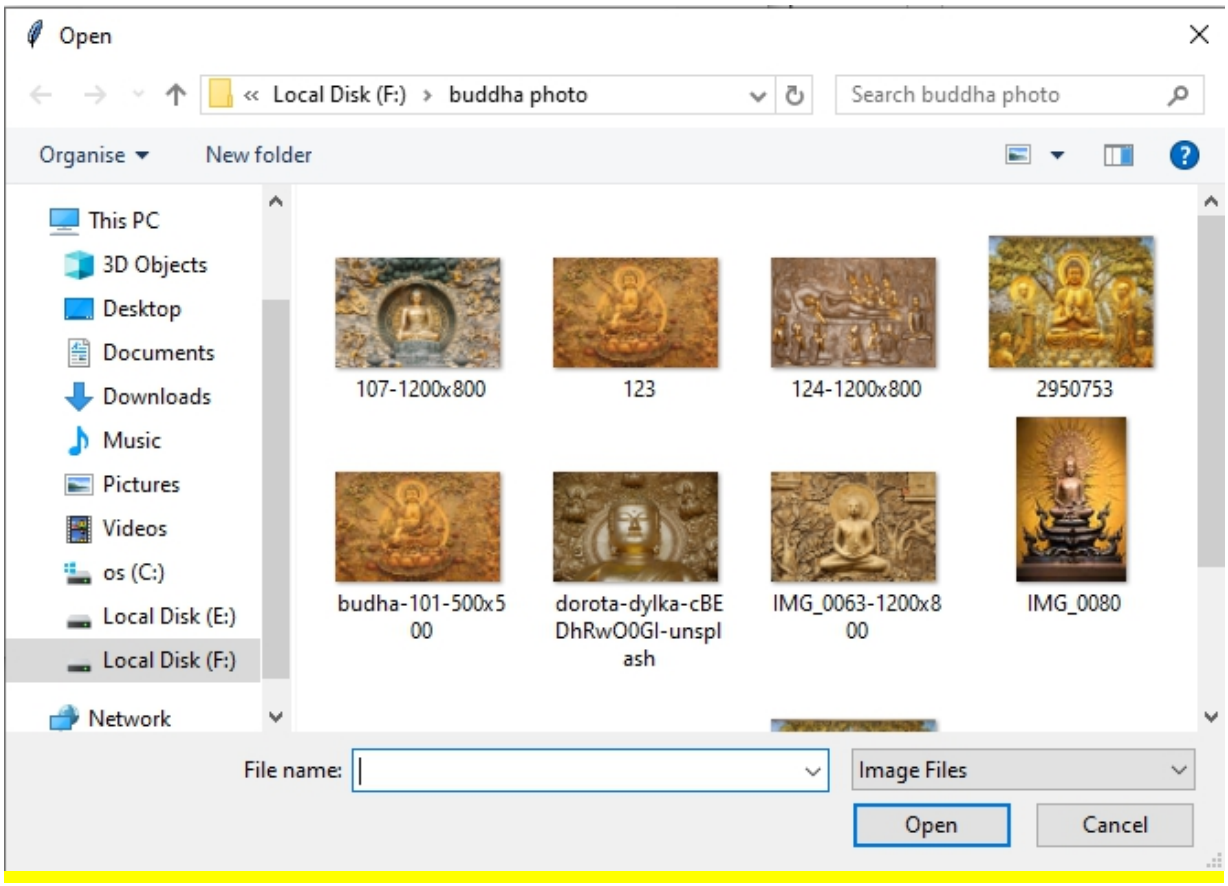
```
cv2.COLOR_BGR2RGB))
    # Convert image to Tkinter format
    original_img_tk = ImageTk.PhotoImage(original_img_pil)
    # Display original image on canvas
    canvas.create_image(0, 0, anchor=tk.NW,
image=original_img_tk)
    # Save reference to avoid garbage collection
    canvas.original_img = original_img_tk
def create_sketch():
    global file_path
    # Check if an image has been opened
    if file_path:
        original_img = cv2.imread(file_path)
        # Convert the original image to sketch
        sketch_img = convert_to_sketch(original_img)
        if sketch_img is not None:
            # Convert sketch image to PIL format
            sketch_img_pil = Image.fromarray(sketch_img)
            # Resize sketch image to fit canvas
            sketch_img_pil = sketch_img_pil.resize((200, 200))
            # Convert sketch image to Tkinter format
            sketch_img_tk =
ImageTk.PhotoImage(sketch_img_pil)
            # Display sketch image on canvas
            canvas.create_image(210, 0, anchor=tk.NW,
image=sketch_img_tk)
            # Save reference to avoid garbage collection
            canvas.sketch_img = sketch_img_tk
    else:
```

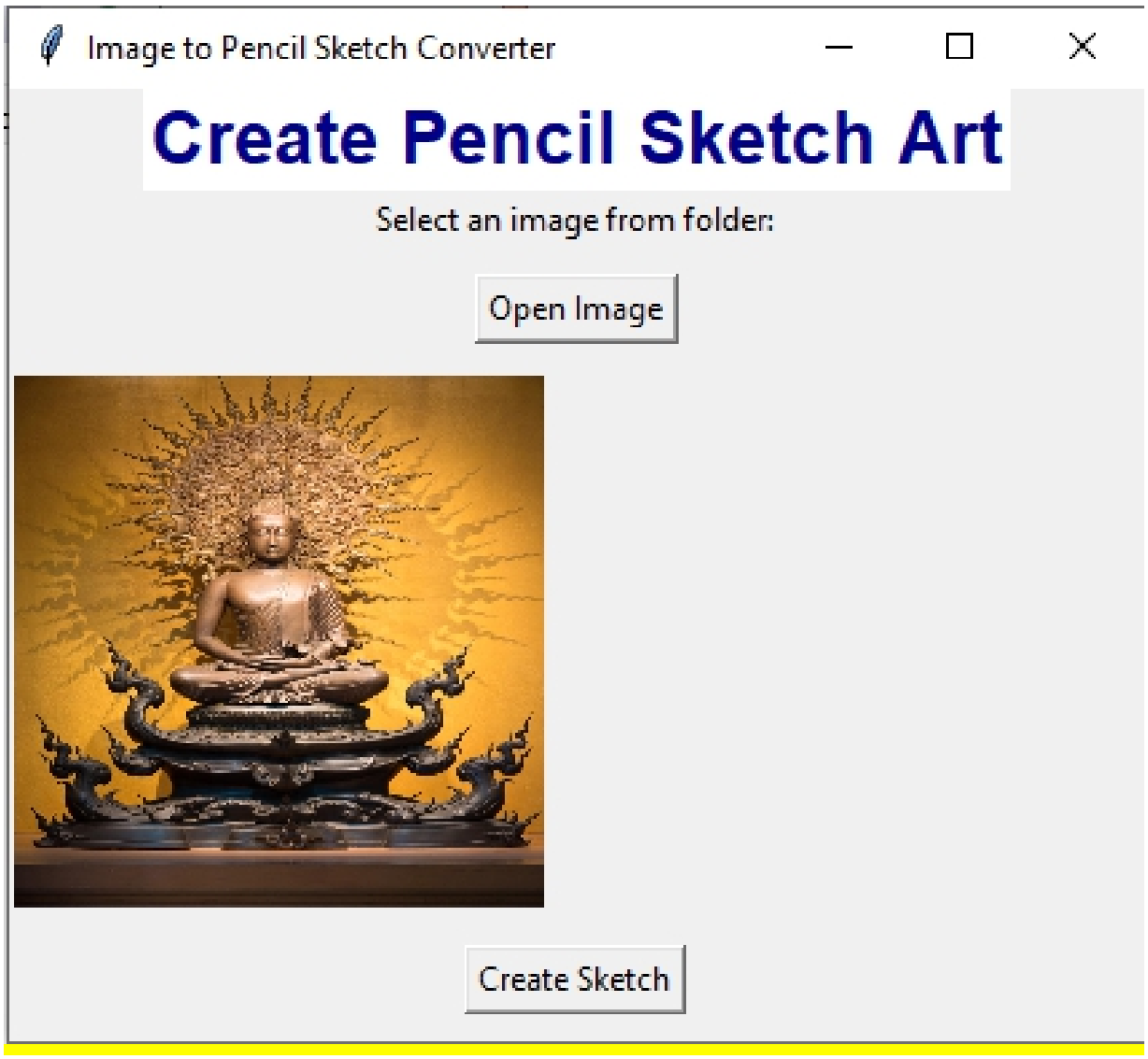
```
        messagebox.showinfo("Info", "Please open an image first.")

root = tk.Tk()
root.title("Image to Pencil Sketch Converter")

# Create a label for the title of the project
title_label = tk.Label(root, text="Create Pencil Sketch Art", fg="dark blue", bg="white", font=("Helvetica", 20, "bold"))
title_label.pack()

# Create a label for selecting an image
label = tk.Label(root, text="Select an image from folder:")
label.pack()

# Create buttons
open_button = tk.Button(root, text="Open Image", command=open_file)
open_button.pack(pady=10)

# Create canvas for displaying images
canvas = tk.Canvas(root, width=420, height=200)
canvas.pack()

sketch_button = tk.Button(root, text="Create Sketch", command=create_sketch)
sketch_button.pack(pady=10)

root.mainloop()
```

Output:

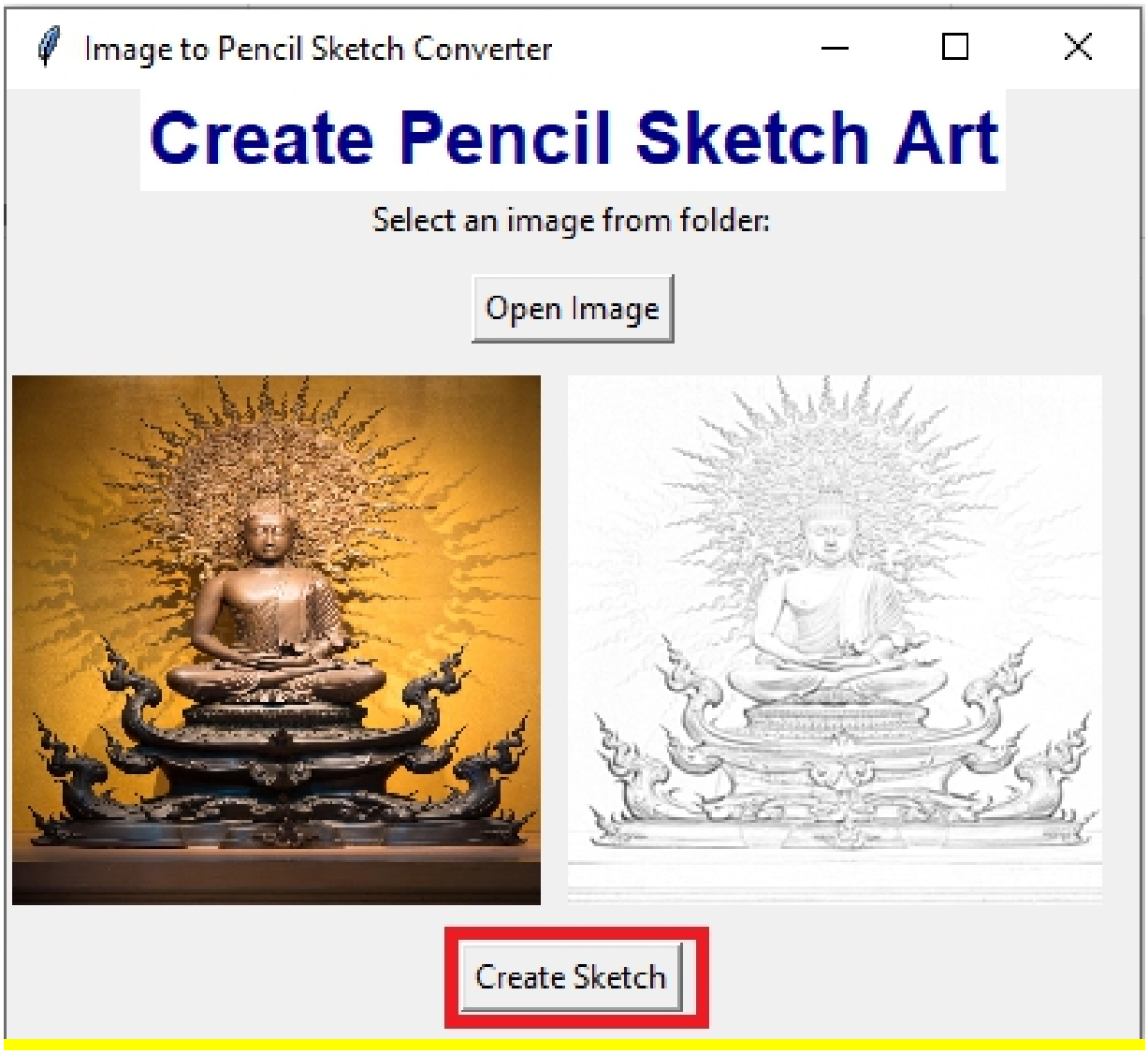


After clicking the 'Open Image' button, the following dialog box is displayed below, allowing the user to select an image."


Open
« Local Disk (F:) › buddha photo
Search buddha photo
Organise
New folder
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
os (C:)
Local Disk (E:)
Local Disk (F:)
Network
107-1200x800
123
124-1200x800
2950753
budha-101-500x500
dorota-dylka-cBEDhRwO0GI-unsplash
IMG_0063-1200x800
IMG_0080
File name:
Image Files
Open
Cancel



After selecting an image from the dialog box, it appears on the left side of the canvas window, positioned above the 'Create Sketch' button, as shown above.



After clicking the 'Create Sketch' button, a pencil sketch image is generated and displayed on the right-hand side of the canvas window.

System Installed Application Finder

Project #15

Develop a Graphical User Interface (GUI) for System Installed application Finder using Tkinter and Python

Introduction:

Welcome to the System Installed App Finder! With this tool, you can easily search for specific apps installed on your computer and access detailed information about them. Simply enter the name of the app you're looking for and click the "Search App" button. The tool will provide you with comprehensive details, including when the app was installed, its version, publisher name, installed location, and uninstall string. If you forget to enter the app name before clicking the button, don't worry! We've implemented a messagebox feature to remind you to input your search query before proceeding.

This information will help you manage and troubleshoot your installed applications effectively. But that's not all! You can also view a list of all apps installed on your computer by clicking the "Display All Apps" button. This feature allows you to see the complete list of installed applications at a glance, making it convenient to review and manage your software.

With the System Installed App Finder, you can easily find the apps you need and access essential information about them in no time.

Requirement :

Modules :

Tkinter : To build the GUI for this project, we'll utilize Tkinter's Entry widget for searching the apps installed in your system, and two Button widgets: one for initiating the search app operation and the other for displaying all installed apps. Additionally, we'll use two

Listboxes: one to display detailed information about the searched app and the other to display details about all installed apps.

Winapps: WinApps is a tool in Python that helps you find out information about the apps you have on your Windows computer. It can tell you things like the name of the app, who made it, when you installed it, and where it's stored on your computer. With WinApps, you can easily check details about your apps, like their versions and when you got them. It's handy for keeping track of what's installed on your system and can help you troubleshoot any issues you might have with your apps.

Using WinApps is straightforward and can be useful for anyone who wants to know more about the software they have on their Windows PC. Whether you're curious about your apps or need to manage them better, WinApps can help you do it easily.

Functions:

list_installed(): You can retrieve a list of all the applications installed on your system. The library provides functions like list_installed or get_apps to fetch this information.

Find_app() : If you're looking for a particular application, you can use the find_app function. This function takes the application name as input and returns details like the application name and its unique identifier.

In this project, we will use list_installed() to fetch the list of all apps installed in our computer. Let's start exploring your installed applications together!

Code:

```
import tkinter as tk

from tkinter import messagebox

import winapps
```

```
def search_apps():
    # Clear previous search results
    listbox1.delete(0, tk.END)
    # Get the search query from the entry widget
    query = entry.get().lower()
    # Check if the search query is empty
    if not query.strip():
        messagebox.showinfo("Empty Search", "Please enter a search query.")
        return
    # Search for applications matching the query
    for app in winapps.list_installed():
        if query in app.name.lower():
            # Display app information in the listbox
            listbox1.insert(tk.END, f"Name: {app.name}")
            listbox1.insert(tk.END, f"Version: {app.version}")
            listbox1.insert(tk.END,f"Publisher: {app.publisher}")
            listbox1.insert(tk.END,f"Installed Location: {app.install_location}")
            listbox1.insert(tk.END, f"Uninstall String: {app.uninstall_string}")
            listbox1.insert(tk.END, "")  # Add an empty line after each app's information
```

```
def display_all_apps():
    # Clear previous search results
    listbox2.delete(0, tk.END)
    # Get all installed applications
    installed_apps = winapps.list_installed()
    # Display all installed applications in the listbox
    for app in installed_apps:
        # Display app information in the listbox
        listbox2.insert(tk.END, f"Name: {app.name}")
        listbox2.insert(tk.END, f"Version: {app.version}")
        listbox2.insert(tk.END, f"Publisher: {app.publisher}")
        listbox2.insert(tk.END, f"Installed Location: {app.install_location}")
        listbox2.insert(tk.END, f"Uninstall String: {app.uninstall_string}")
        listbox2.insert(tk.END, "")  # Add an empty line after each app's information

# Create the Tkinter window
root = tk.Tk()
root.title("Installed Apps")
title_label = tk.Label(root, text="System Installed App Finder")
title_label.grid(row=0, column=0, padx=10, pady=10)
```

```
title_label.config(fg="dark red", bg="white", font=
("Helvetica",15,"bold"))

# Create search entry and buttons

entry = tk.Entry(root, width=50)

entry.grid(row=1, column=0, padx=10, pady=10)

# Create search button

search_button = tk.Button(root, text="Search Apps",
command=search_apps)

search_button.grid(row=2, column=0, padx=10, pady=5)

# Create listbox to display search results

listbox1 = tk.Listbox(root, width=50, height=10)

listbox1.grid(row=3, column=0, padx=10, pady=5)

# Create scrollbar for listbox2

scrollbar1 = tk.Scrollbar(root, orient=tk.VERTICAL,
command=listbox1.yview)

scrollbar1.grid(row=3, column=1, sticky="ns")

listbox1.config(yscrollcommand=scrollbar1.set)

# Create horizontal scrollbar for listbox1

scrollbar1x = tk.Scrollbar(root, orient=tk.HORIZONTAL,
command=listbox1.xview)

scrollbar1x.grid(row=4, column=0, sticky="ew")

listbox1.config(xscrollcommand=scrollbar1x.set)
```

```
display_button = tk.Button(root, text="Display All Apps", command=display_all_apps)

display_button.grid(row=5, column=0, padx=10, pady=5)

# Create listbox to display all installed applications

listbox2 = tk.Listbox(root, width=50, height=15)

listbox2.grid(row=6, column=0, padx=10, pady=5)

# Create scrollbar for listbox2

scrollbar2 = tk.Scrollbar(root, orient=tk.VERTICAL, command=listbox2.yview)

scrollbar2.grid(row=6, column=1, sticky="ns")

listbox2.config(yscrollcommand=scrollbar2.set)

# Create horizontal scrollbar for listbox2

scrollbar2x = tk.Scrollbar(root, orient=tk.HORIZONTAL, command=listbox2.xview)

scrollbar2x.grid(row=7, column=0, sticky="ew")

listbox2.config(xscrollcommand=scrollbar2x.set)

# Start main loop

root.mainloop()
```

Output :

After executing the provided code, the GUI for the System Installed App Finder will display as follows:

Title: At the top of the GUI, you'll find the title positioned at the center, labeled "System Installed App Finder" in a large font.

Entry Box: Below the title, there is entry box which allows users to input the name of the app they want to find. They can type the app name into this box.

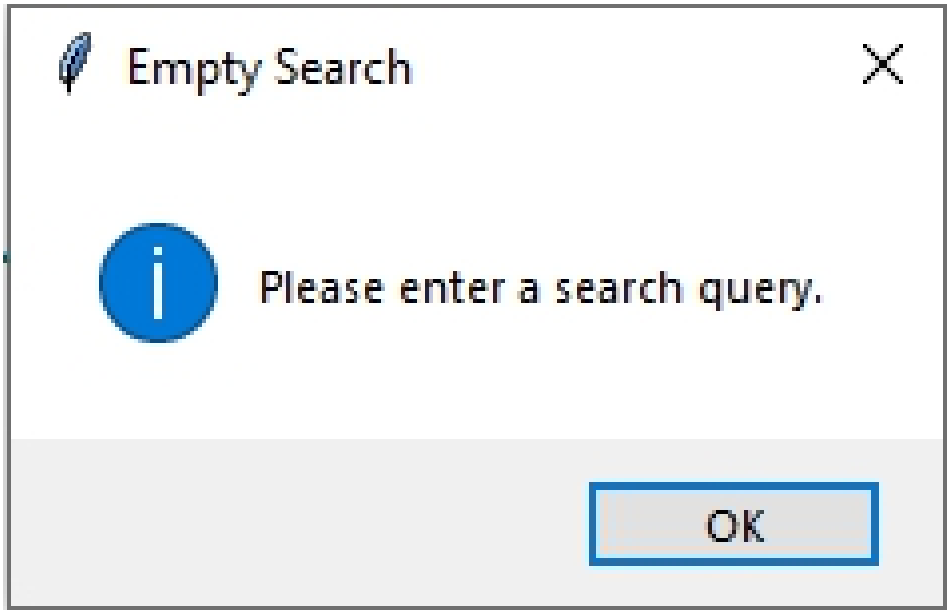
Search Apps Button: Following the entry box, a button labeled "Search Apps" prompts users to initiate the search process for the entered app name.

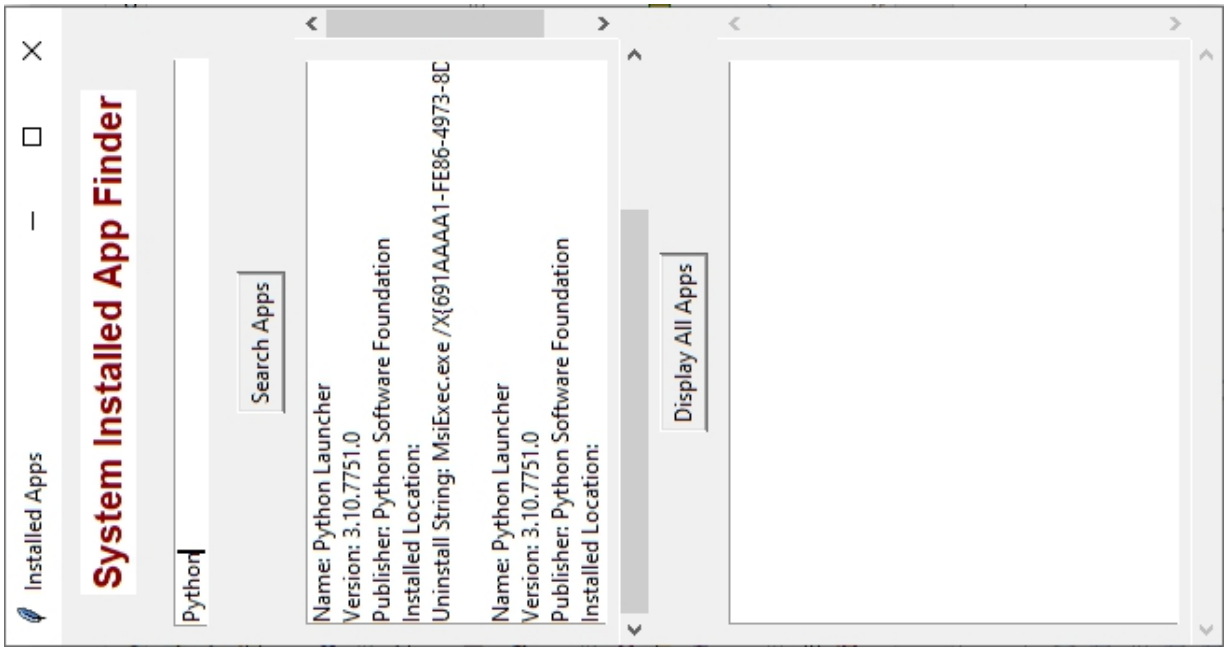
Listbox (First): Below the button,you will find the listbox that displays all the details about the searched app, such as app name,installation date, version, publisher, etc. Horizontal and vertical scrollbars are applied to the listbox for easy navigation if the details exceed the listbox size.

Display All Apps Button: Beneath the first listbox, users find a button titled "Display All Apps", allowing them to view details of all installed apps in a separate listbox.

Listbox (Second): This listbox shows the details of all installed apps when the "Display All Apps" button is clicked. Similar to the first listbox, it also has horizontal and vertical scrollbars for easy viewing.

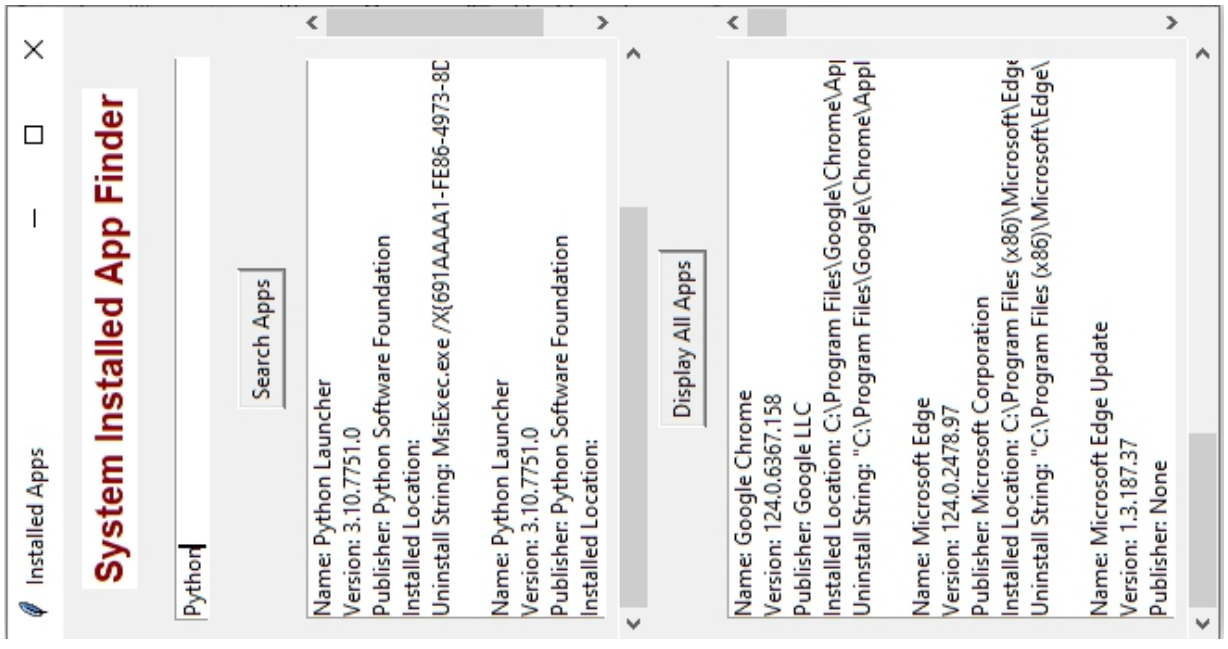
The user will see the above GUI after executing the code. If the user does not type any app name and clicks on the search apps button, the following error message is displayed: "Please enter a search query."





Here, when the user enters "python" in the entry box and

clicks on the search apps button, all details about the Python app are displayed in the listbox as shown above.



When the user clicks the "Display All Apps" button, all the apps installed on their system are shown, as depicted above.

All-in-one Temperature Convertor

Project #16

Develop a Python GUI application for All-in-one Temperature Convertor using Tkinter.

Introduction:

Welcome to the Temperature Converter! This handy tool helps you change temperatures between Fahrenheit, Celsius, and Kelvin easily. Whether you need to convert from Fahrenheit to Celsius, Celsius to Fahrenheit, Kelvin to Celsius, or any other combination, this tool has got you covered. Just pick the units you're converting from and to, type in the number, and hit "Convert." It's that simple! Our goal with this project is to offer you a convenient and user-friendly way to perform temperature conversions with ease. So, whether you're a student, scientist, or simply someone who needs to convert temperatures on a regular basis, let the Universal temperature Unit Converter simplify your tasks. Let's get started and make those conversions!

Before learning the temperature conversion formulas, let us recall the units of temperature in short.

Units of Temperature

Different units can be used to record the temperature. The three different units used for measuring temperature are Celsius (°C) Fahrenheit (°F), and Kelvin (K). Kelvin is the SI unit of measuring temperature, whereas Fahrenheit and Celsius are commonly used scales.

Temperature Conversion Formulas:

- Celsius to Kelvin: K = C + 273.15
- Kelvin to Celcius: C = K - 273.15
- Fahrenheit to Celcius: C = (F-32) (5/9)
- Celsius to Fahrenheit: F = C(9/5) + 32

- Fahrenheit to Kelvin: K = (F-32) (5/9) + 273.15
- Kelvin to Fahrenheit: F = (K-273.15) (9/5) + 32

Example: Convert 20°C into Kelvin.

K = C + 273.15

= 20 + 273.15

= 293.15 K

Therefore, 20°C is equivalent to 293.15 K.

Example : Convert 120°F to Celsius

F = 120°F.

Using Fahrenheit to Celsius conversion formula,

C = (F − 32) × 5/9

= (120 − 32) × (5/9)

= 48.88°C

Therefore, 115°F is 48.88°C on the centigrade scale.

To create the GUI for this project, we'll use two entry fields for inputting the "from" and "to" temperature values. Additionally, we'll include two dropdown menus for selecting the temperature units for conversion, along with a convert button to initiate the conversion process. The result will be displayed in the "to" entry box. Lets try to build this form together!

Lets try to implement the code now.

Code :

```
import tkinter as tk

def convert_temperature():
```

```
try:
    # Get input value
    value = float(entry_value.get())
    # Convert to desired unit
    if var_from.get() == "Fahrenheit":
        if var_to.get() == "Celsius":
            result = (value - 32) * 5/9
        elif var_to.get() == "Kelvin":
            result = (value - 32) * 5/9 + 273.15
    elif var_from.get() == "Celsius":
        if var_to.get() == "Fahrenheit":
            result = (value * 9/5) + 32
        elif var_to.get() == "Kelvin":
            result = value + 273.15
    elif var_from.get() == "Kelvin":
        if var_to.get() == "Fahrenheit":
            result = (value - 273.15) * 9/5 + 32
        elif var_to.get() == "Celsius":
            result = value - 273.15
    # Clear any previous result
    entry_result.delete(0, tk.END)
```

```
        # Insert new result
        entry_result.insert(tk.END, str(result))
    except ValueError:
        entry_result.delete(0, tk.END)
        entry_result.insert(tk.END, "Invalid input")
# Create main window
root = tk.Tk()
root.title("Temperature Converter")
# Create Label for Title
label_title = tk.Label(root, text="Universal Temperature Unit Convertor", fg="white", bg="dark blue")
label_title.grid(row=0, column=1, columnspan=3, pady=10)
label_title.config(font=("Helvetica", 15, "bold"))
# Create input entry
entry_value = tk.Entry(root)
entry_value.grid(row=2, column=1)
# Create "From" unit dropdown
var_from = tk.StringVar(root)
var_from.set("Fahrenheit")
option_from = tk.OptionMenu(root, var_from, "Fahrenheit", "Celsius", "Kelvin")
option_from.grid(row=3, column=1)
```

 # Create "=" label

```
label_eq = tk.Label(root, text= " =")

label_eq.grid(row=3, column=2)

label_eq.config(font=("Helvetica", 20, "bold"))
```

 # Create input entry

```
entry_result = tk.Entry(root)

entry_result.grid(row=2, column=3)
```

 # Create "To" unit dropdown

```
var_to = tk.StringVar(root)

var_to.set("Celsius")

option_to = tk.OptionMenu(root, var_to, "Celsius", "Fahrenheit",
"Kelvin")

option_to.grid(row=3, column=3, pady=5)
```

 # Create button to trigger conversion

```
button_convert = tk.Button(root, text="Convert",
command=convert_temperature)

button_convert.grid(row=6, column=1, columnspan=3)

button_convert.config(width=10)
```

 # Run the GUI

```
root.mainloop()
```

Output:

After executing the provided code, you'll see the GUI for the Universal Temperature Unit Converter.

Title : At the top of the window, there's a title labeled "Universal Temperature Unit Converter." Underneath the title, there are two textboxes side by side.

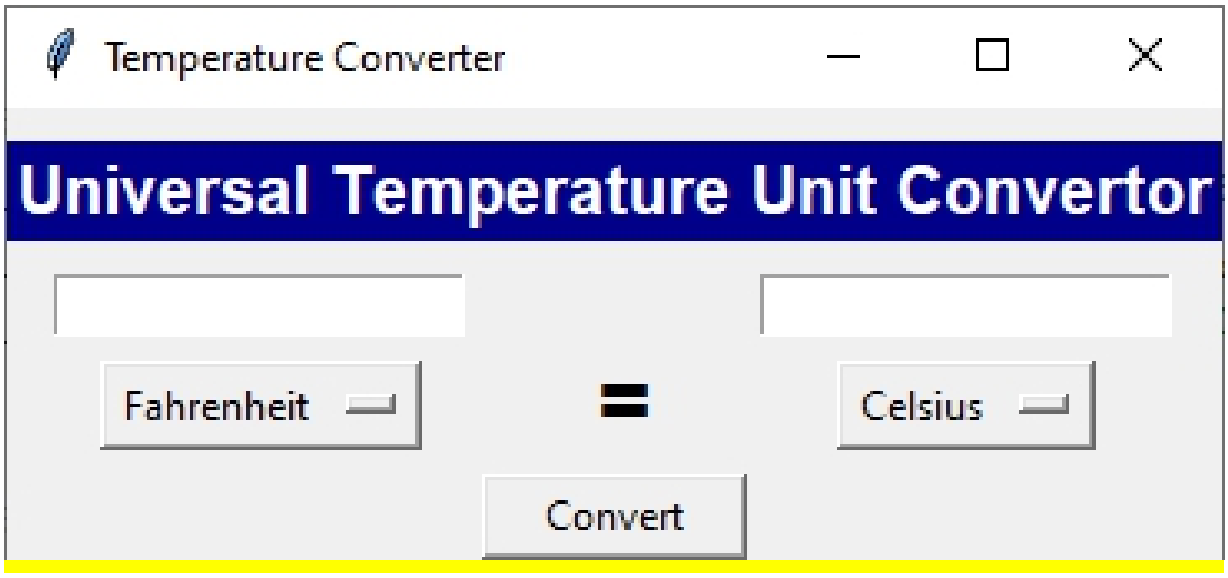
Input Textbox: Positioned on the left side of the GUI, this textbox allows users to input numerical values representing temperatures. Users can type the temperature value they want to convert into this textbox.

Input Unit Combobox: Below the input textbox, there is a combobox providing a dropdown menu of options for selecting the unit of temperature (e.g., Fahrenheit, Celsius, Kelvin). Users can choose the appropriate unit from this combobox for the input temperature.

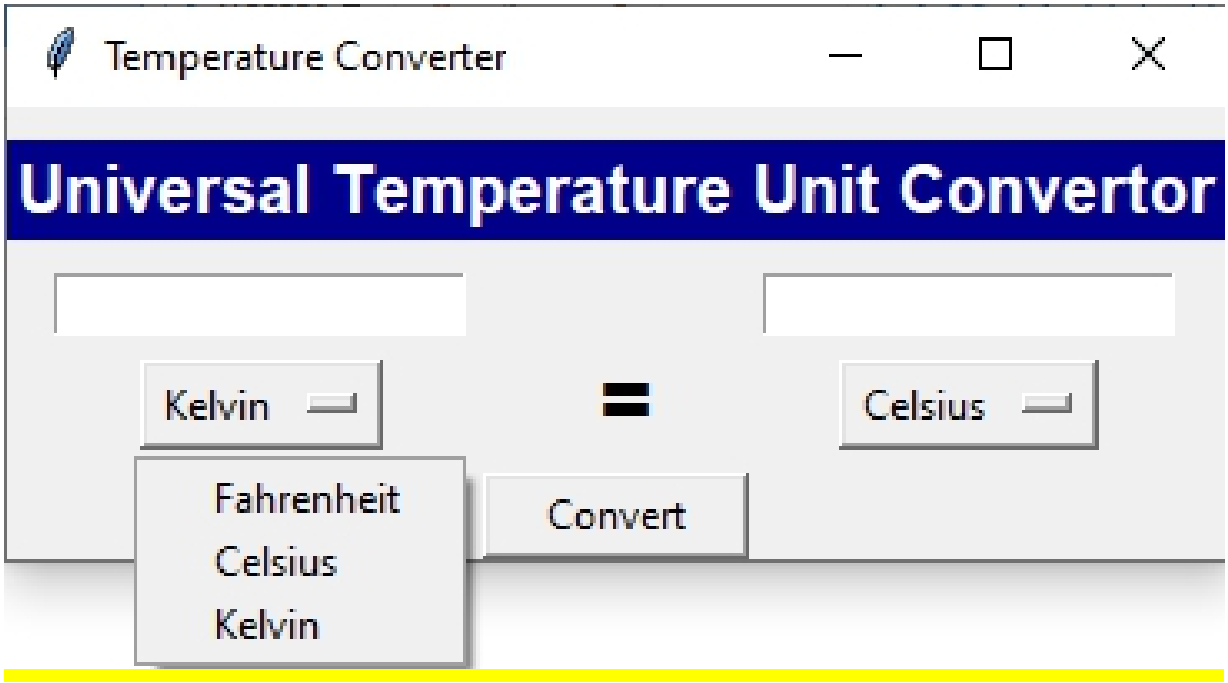
Result Textbox: Positioned on the right side of the GUI, this textbox displays the converted temperature value after the conversion process is triggered.

Output Unit Combobox: Below the result textbox, there is another combobox providing options for selecting the unit of temperature for the converted value.

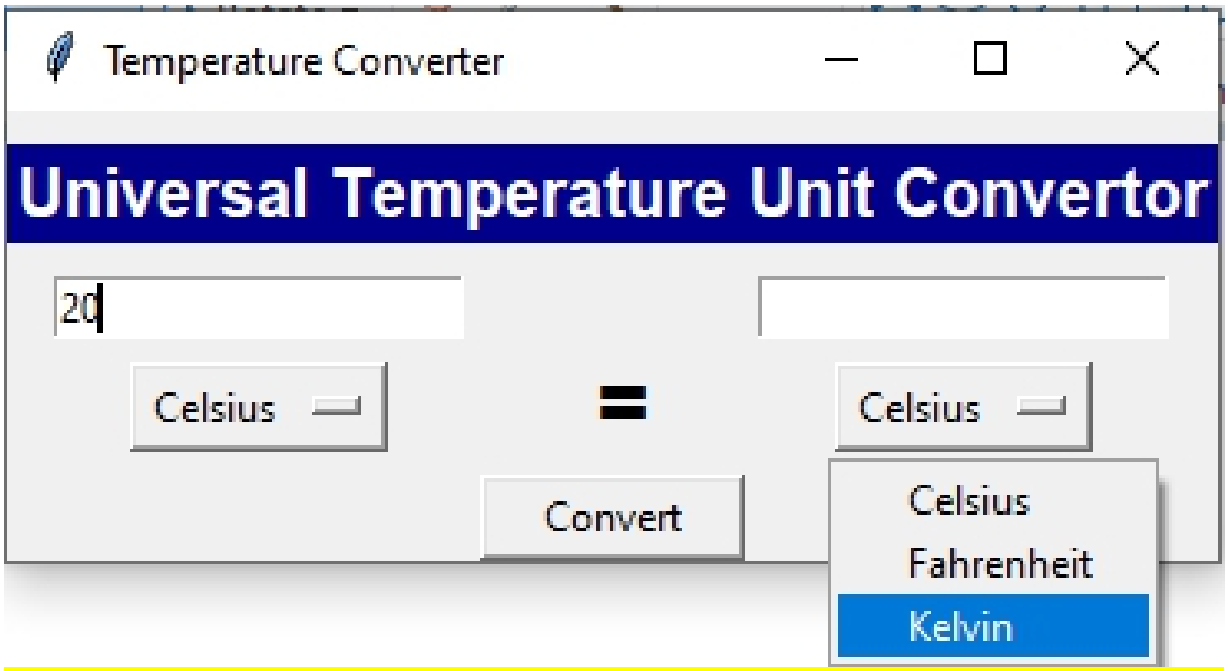
Convert Button: At the center of window ,below both comboboxes, there is a "Convert" button. Clicking this button triggers the conversion process. After entering the temperature value and selecting the unit, users can click this button to perform the conversion, and the result will be displayed in the result textbox.



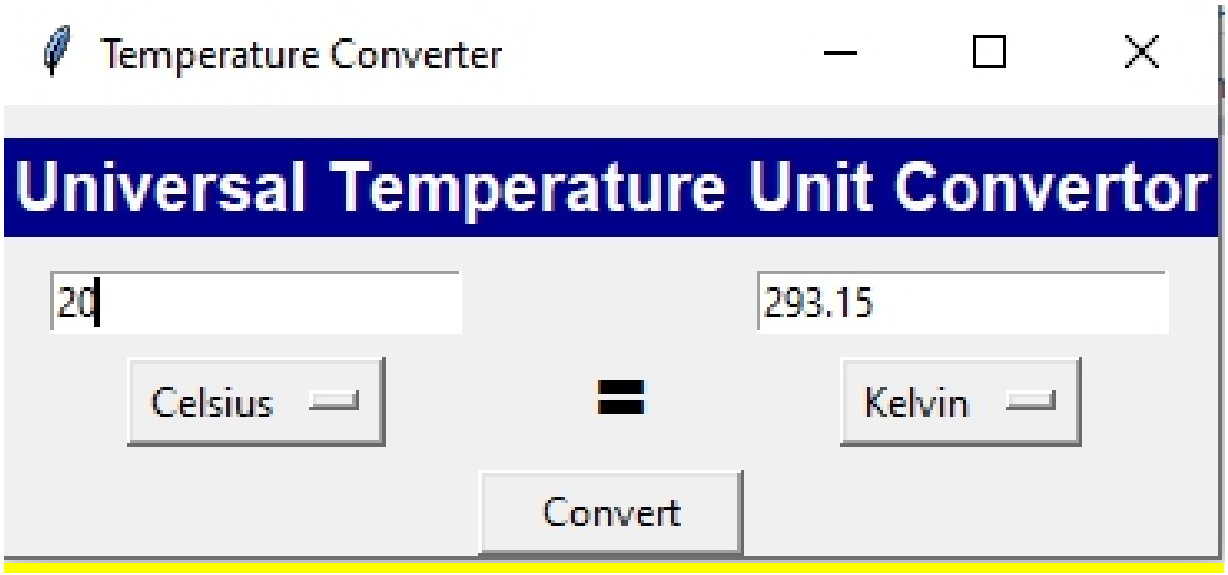
The user will see the above GUI after running the code.



The user can select the temperature from the dropdown box. In this example, Kelvin is selected for conversion.



Here, Celsius is selected for conversion and then a value is entered. The user wants to convert 20 Celsius to Kelvin, so they select Kelvin as the output temperature.



When the user clicks the convert button, the converted result is displayed in the result output box, as shown above.

QR Code Scanner

Project #17

Design python GUI for QR Code Scanner using tkinter

Introduction

Welcome to our QR Code Extractor! Have you ever been curious about what's hidden inside those little square codes? Well, wonder no more! Our tool is here to make decoding QR codes easy and fun. The aim of our project is simple: to provide a hassle-free way for users to extract information from QR code images.

With just a few clicks, you can upload a QR code image using the "Browse" button and see it displayed right on your screen. Then, with another click on "Extract Data," our tool goes to work, decoding the QR code and revealing all the important information it contains. Whether it's a website link, contact details, or something else entirely, our QR Code Extractor saves you time and effort by automating the process. No more manual transcription required!

Whether you're a tech enthusiast, a business professional, or just curious about what's hidden in those codes, our tool is designed to make decoding QR codes simple and accessible for everyone.

Let's start unlocking secrets together!

Requirement :

Modules used:

The cv2 module, also known as OpenCV, is a powerful library used for computer vision tasks in Python. "CV" stands for computer vision, and "2" denotes that it's the second version of the OpenCV library for Python.

The cv2 module, also called OpenCV, is a handy tool for working with images and videos in Python. It helps with tasks like changing the size of images, cutting them into pieces, turning them around, making them blurry or sharp, and much more. You can also use cv2 to work with videos, like watching them in real-time or making changes to them. This could mean anything from playing video files to taking pictures with a webcam and applying different effects to them.

Methods of cv2 modules:

Here's an overview of some commonly used methods and functions:

Image Reading and Writing:

cv2.imread(): Read an image from a file.

cv2.imwrite(): Write an image to a file.

Image Processing:

cv2.resize(): Resize an image to a specified size.

cv2.cvtColor(): Convert an image from one color space to another (e.g., RGB to grayscale).

cv2.threshold(): Apply a fixed-level threshold to an image.

cv2.blur(), cv2.GaussianBlur(), cv2.medianBlur(): Apply various types of blurring to an image.

Video Processing:

cv2.VideoCapture(): Open a video file or capture device (e.g., webcam) for reading.

cv2.VideoWriter(): Create a video file for writing.

Drawing and Visualization:

cv2.line(), cv2.rectangle(), cv2.circle(): Draw lines, rectangles, and circles on an image.

cv2.putText(): Write text on an image.

cv2.polylines(): Draw polylines (i.e., sequences of connected straight lines) on an image.

Utilities:

cv2.waitKey(), cv2.destroyAllWindows(): Wait for a key press and destroy windows.

These are just a few examples of the many methods and functions provided by the cv2 module for various computer vision tasks.

To use the cv2 module for QR code scanning in Python, you'll need to install OpenCV :

If you haven't already installed OpenCV, you can do so using pip:

pip install opencv-python

Pyzbar module: Pyzbar is a Python library that allows you to decode barcodes and QR codes from images using the ZBar library. It allow user to decodes various barcode formats including QR codes, linear barcodes (UPC-E, EAN, Code 39, etc.)

It can Works with different image formats: PIL/Pillow images, OpenCV images, raw image bytes and provides decoded data (text content)

You can install pyzbar using pip:

pip install pyzbar

Steps to implement this project:

To display the image in an image label, we'll follow these steps:

- Read the QR code image using the cv2.imread() function.
- Convert the image to the RGB format using cv2.cvtColor().
- Convert the image to the PIL format using Image.fromarray().
- Create a PhotoImage object from the PIL image using ImageTk.PhotoImage().

To extract data from the QR code image, we'll do the following:

- Read the image using cv2.imread().
- Convert it to grayscale using cv2.cvtColor().
- Find the QR code in the grayscale image using pyzbar.decode().
- Extract the data.

Lets see how to implement this steps in our code

Code:

```
import tkinter as tk

from tkinter import filedialog

from PIL import Image, ImageTk

import cv2

from pyzbar import pyzbar

# IMage Browse function

def browse_qr_image():

    # Open file dialog box to select QR code image

    filename = filedialog.askopenfilename(filetypes=[("Image files", "*.png;*.jpg;*.jpeg")])

    # if it is open then call show_image() function
```

```
    if filename:
        show_image(filename)
# Display image function
def show_image(filename):
    # Store the filename globally
    global img_filename
    img_filename = filename
    # Read the QR code image
    qr_image = cv2.imread(filename)
    # Convert the image to RGB format
    rgb_image = cv2.cvtColor(qr_image, cv2.COLOR_BGR2RGB)
    # Convert the image to PIL format
    pil_image = Image.fromarray(rgb_image)
    # Create a PhotoImage object from the PIL image
    img = ImageTk.PhotoImage(pil_image)
    # Display the image in the image_label widget
    image_label.config(image=img)
    image_label.image = img
# Extract QR code data function
def extract_qr_data():
    # Check if an image is selected
    if img_filename:
        # Read the QR code image
        qr_image = cv2.imread(img_filename)
```

```
        # Convert the image to grayscale
        gray_image = cv2.cvtColor(qr_image,
cv2.COLOR_BGR2GRAY)

        # Find QR codes in the grayscale image
        qrcodes = pyzbar.decode(gray_image)

        if qrcodes:
            # Extract data from the QR code
            qr_data = qrcodes[0].data.decode("utf-8")

            # Display the extracted data
            result_text.delete(1.0, tk.END)
            result_text.insert(tk.END, "QR Code Data: " + qr_data)
        else:
            result_text.delete(1.0, tk.END)
            result_text.insert(tk.END, "No QR code found in the
image.")
    else:
        result_text.delete(1.0, tk.END)
        result_text.insert(tk.END, "Please select a QR code image
first.")

# Create main window
root = tk.Tk()
root.title("QR Code Data Extractor")

# Create Label for Title
label_title = tk.Label(root, text="QR Code Extractor", fg="White",
bg="Navy Blue")
label_title.grid(row=0, column=0, columnspan=3, pady=5)
label_title.config(font=("Helvetica", 25, "bold"))

# Create Label for button
label_title = tk.Label(root, text="Upload QR Code Image", fg="Dark
blue")
label_title.grid(row=1, column=0, columnspan=3, pady=5)
label_title.config(font=("Helvetica", 10, "bold"))
```

```
# Initialize global variable for storing filename
img_filename = None

# Create button to browse QR code image
browse_image_button = tk.Button(root, text="Browse",
command=browse_qr_image)
browse_image_button.grid(row=2, column=0, columnspan=3)

# Create label to display image
image_label = tk.Label(root)
image_label.grid(row=3, column=0, columnspan=3)

# Create button to extract data
extract_data_button = tk.Button(root, text="Extract Data",
command=extract_qr_data)
extract_data_button.grid(row=4, column=0)

# Create text widget to display extracted data
result_text = tk.Text(root, height=5, width=50)
result_text.grid(row=5, column=0, pady=10)

root.mainloop()
```

Output:

In the GUI of the QR code scanner, you'll see:

Title: At the top of the GUI, there's a title labeled "QR Code Scanner."

Instructional Label: Under the title, there's an instructional label prompting the user to upload a QR code image.

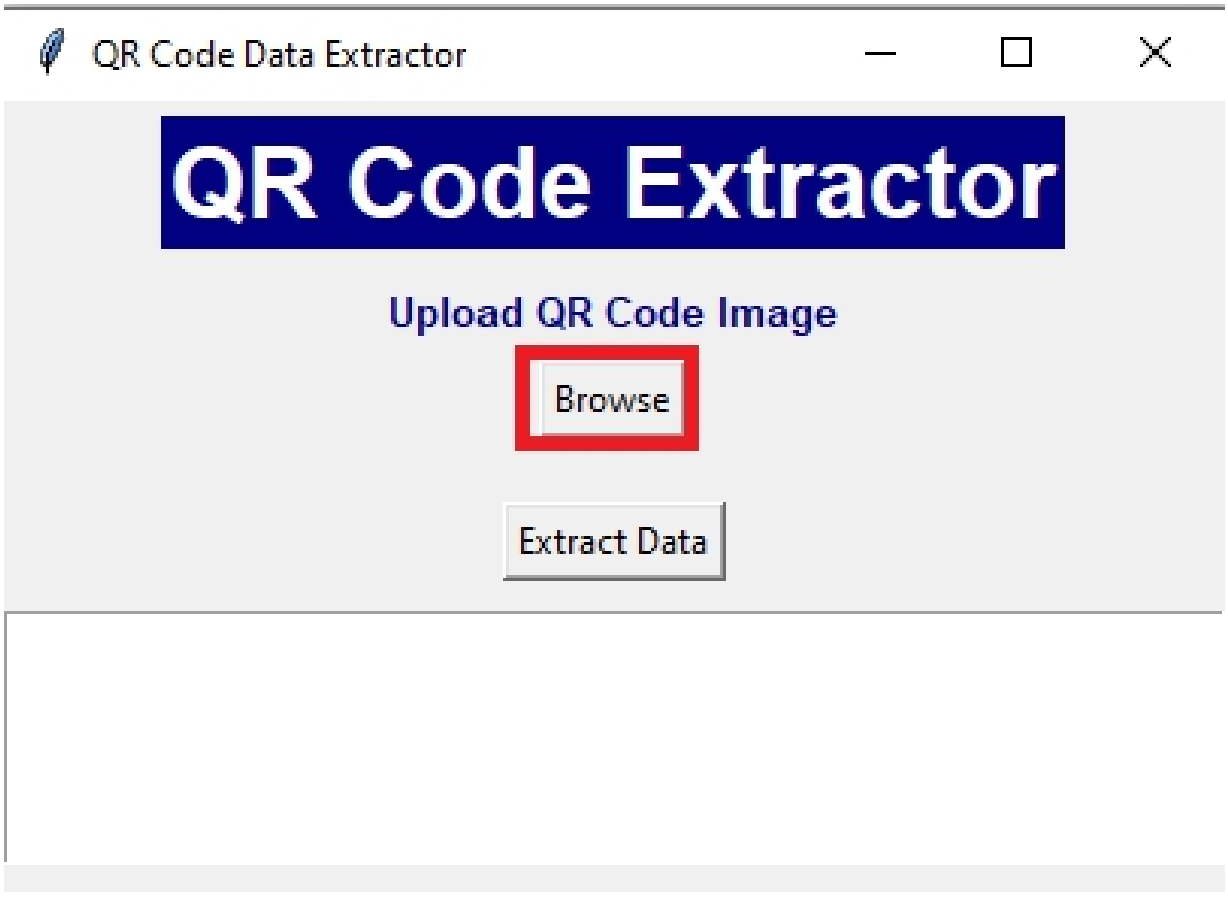
Browse Button: Below the instructional label, there's a "Browse" button. Clicking this button allows the user to browse and select a QR code image from their device.

Image Display Area: Upon selecting a QR code image using the "Browse" button, the selected image is displayed in a label on the

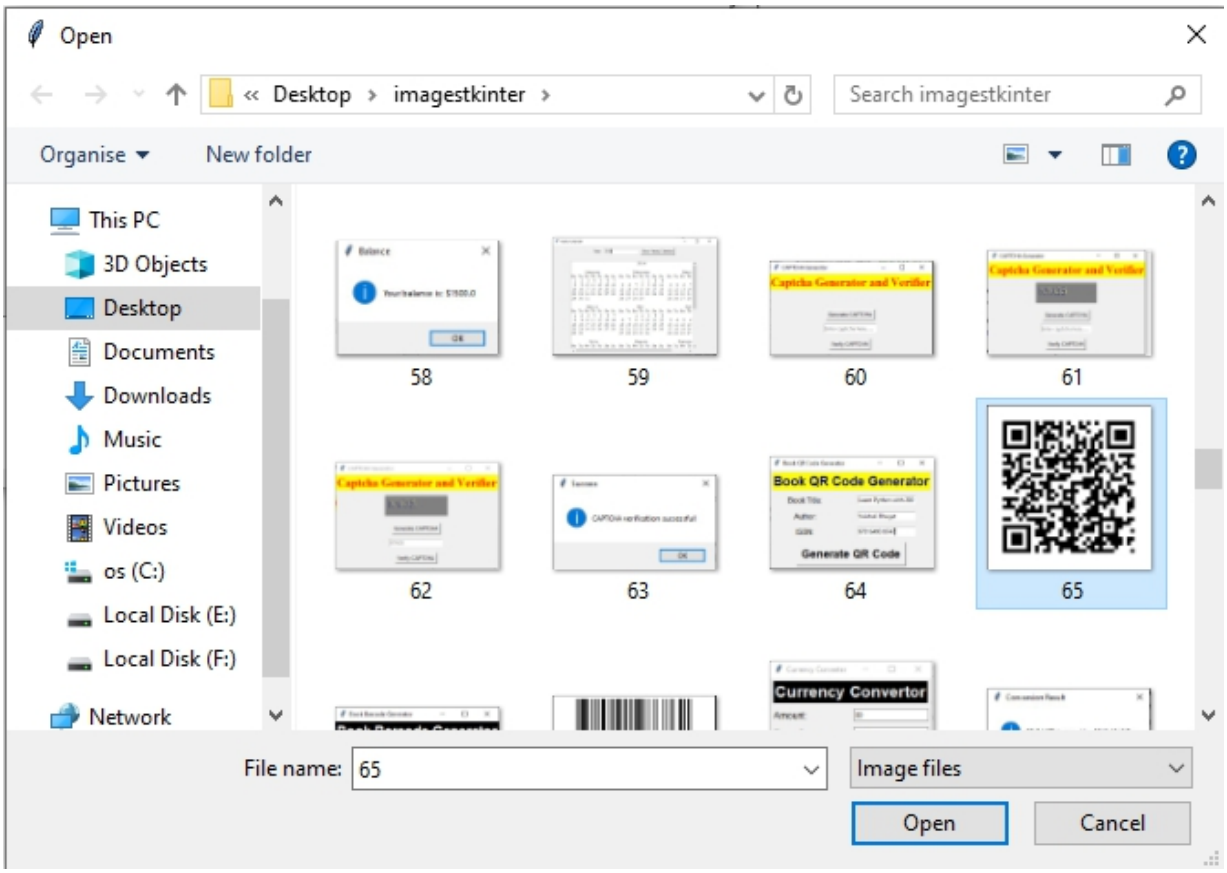
window.

Extract Data Button: Under the image label, there's an "Extract Data" button. Clicking this button triggers the extraction process of the QR code data.

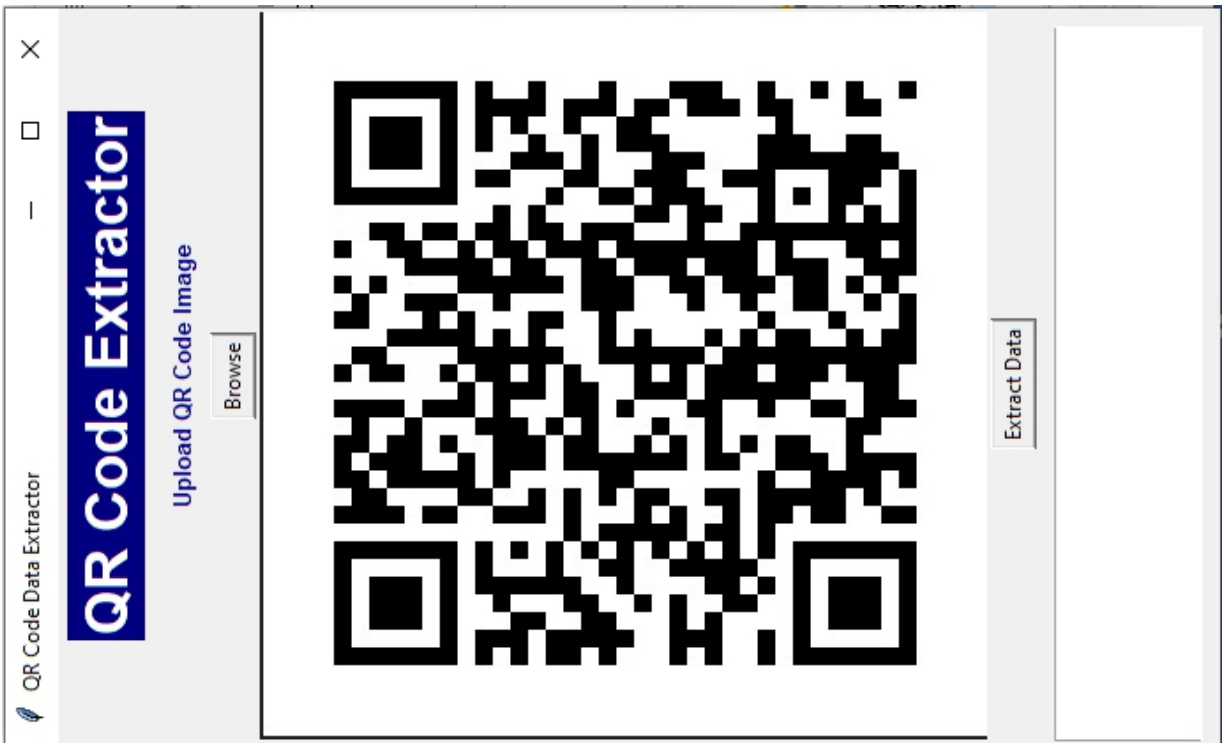
Text Area: Once the data is extracted, the decoded QR code content is displayed in a text area provided on the main window. Users can view the extracted code here.



This is the main GUI window, where users can upload a QR code image by clicking on the browse button.



The above open file dialogue box is opened, and the user can select the QR code image that they want to scan.



The QR code image is automatically uploaded beneath the browse button, as depicted above.

After clicking on the "Extract Data" button, all data such as book title, author name, and ISBN number are extracted from the selected QR code image and displayed beneath the button in the text box.

Simple Slideshow Application

Project #18

Design Python GUI for Slideshow Creator Application using Tkinter

Introduction:

Welcome to the Slideshow App! It's a handy tool for showing off your photos in a slideshow. With this app, you can easily upload a bunch of photos from a folder. Once they're uploaded, a message will pop up saying your photos are ready to go.

Just hit the "Start" button to kick off the slideshow. Your photos will appear one by one, with a little break of 2 seconds between each. The slideshow plays right on top of all the buttons, so it's easy to see.

If you need to take a break or want to pause the slideshow, just hit the "Pause" button. You can start it up again whenever you're ready by hitting "Resume." And if you're all done, just hit "Stop" to end the slideshow. Lets build this amazing slideshow application together.

Requirement:

Modules used:

tkinter: tkinter is a standard GUI toolkit for Python. It provides various widgets (such as buttons, labels, and frames) that you can use to build interactive applications.

filedialog: The filedialog module in tkinter allows you to create dialog boxes for file and directory selection. In this project, it will be used specifically for opening a folder dialog to choose the image folder.

messagebox: The messagebox module provides functions for displaying various types of message boxes, including information, warning, error, and question boxes. In this project, it will be used to inform the user that the image folder has been successfully uploaded.

os: The os module is part of the Python Standard Library.Modules included in the Standard Library come pre-installed with Python, so you can import and use them directly without any additional installation steps. It means you do not need to use pip to install the os module.

For example, you can simply import the os module and use its functions as shown below:

import os

The os module provides a way to interact with the operating system, allowing you to perform various tasks such as file manipulation, directory traversal, and environment variables management. In this project, it will be used to work with file paths and directory operations.

PIL (Python Imaging Library): PIL (now known as Pillow) is a library for opening, manipulating, and saving many different image file formats. It provides functions for tasks such as resizing, cropping, rotating, and converting images between different formats. In this project, it will be used to handle image processing tasks.(As we discussed this module many times in this book.)

Image and ImageTk:

These classes are part of the PIL (Pillow) library and provide functionality for working with images in Python. Image is used to open and manipulate images, while ImageTk is used to convert images into a format that can be displayed in a tkinter GUI. In this project, they will be used to load and display images in the slideshow.

time: The time module provides functions for working with time-related tasks, such as measuring time intervals and delaying program execution. In this project, it will be used to control the timing between image transitions in the slideshow.

The time module is a part of the Python Standard Library, so you don't need to install it separately. It come pre-installed with Python, which means you can use it without any additional installation steps.

Code:

```
import tkinter as tk

from tkinter import filedialog, messagebox

import os

from PIL import Image, ImageTk

import time

slideshow_running = False  # Flag to indicate if slideshow is running

paused = False  # Flag to indicate if slideshow is paused

images = [] # List to store loaded images

current_image_index = 0  # Index of the currently displayed image

# Load images function

def load_images():

    global images

    # Open file dialog to select a folder

    folder_path = filedialog.askdirectory()
```

```
    # Check if a folder was selected
    if folder_path:
        images = []
        # Iterate over files in the selected folder
        for file in os.listdir(folder_path):
            # Check if the file has a valid image extension
            if file.endswith(('jpg', 'jpeg', 'png')):
                # Construct full path of the image file and add it to the list
                images.append(os.path.join(folder_path, file))
        # Show message box to confirm images are loaded
        messagebox.showinfo("Success", "Image folder loaded successfully!")
# Function to start slide show
def start_slideshow():
    global slideshow_running, paused, current_image_index
    if images:
        slideshow_running = True
        for i in range(current_image_index, len(images)):
            if not slideshow_running:
                break  # Stop slideshow if flag is set to False
            if paused:
```

```
            # If paused, schedule the slideshow to resume after a delay
            root.after(1000, start_slideshow)
            return
        current_image_index = i
        display_image(images[i])
        # Adjust delay between slides (2 sec)
        time.sleep(2)
# Image display function
def display_image(image_path):
    # Open and resize image
    image = Image.open(image_path)
    image = image.resize((400, 300), Image.LANCZOS)
    # Convert image to PhotoImage
    photo = ImageTk.PhotoImage(image)
    # Display image on image label
    image_label.config(image=photo)
    image_label.image = photo
    # Update GUI
    root.update()
# Function to stop slideshow
def stop_slideshow():
```

```
    global slideshow_running, paused
    slideshow_running = False
    paused = False
# Function to pause slideshow
def pause_slideshow():
    global paused
    paused = True
# Function to resume slideshow
def resume_slideshow():
    global paused
    paused = False
    start_slideshow()
root = tk.Tk()
root.title("Slideshow App")
# Create a label for title of project
title_label = tk.Label(root, text="Slideshow App")
title_label.pack()
title_label.config(fg="dark red", bg="white", font=
("Helvetica",20,"bold"))
# Create label for displaying images during slideshow
image_label = tk.Label(root)
image_label.pack()
```

 # Create load Image button

```
load_button = tk.Button(root, text="Load Image Folder", command=load_images)

load_button.pack(side=tk.LEFT)
```

 # Create start slideshow button

```
start_button = tk.Button(root, text="Start Slideshow", command=start_slideshow)

start_button.pack(side=tk.LEFT)
```

 # Create stop slideshow button

```
stop_button = tk.Button(root, text="Stop Slideshow", command=stop_slideshow)

stop_button.pack(side=tk.LEFT)
```

 # Create pause slideshow button

```
pause_button = tk.Button(root, text="Pause Slideshow", command=pause_slideshow)

pause_button.pack(side=tk.LEFT)
```

 # Create resume slideshow button

```
resume_button = tk.Button(root, text="Resume Slideshow", command=resume_slideshow)

resume_button.pack(side=tk.LEFT)

root.mainloop()
```

Output :

After executing above provided code, you will see

Title Label: At the top of the GUI, there is a centered title label that reads "Slideshow App". This label serves as the main heading for the application.

Instructional Label: Directly below the title label, there is an instructional label that reads "Select Image Folder". This label provides guidance to the user on what action to take next.

Resultant Slideshow: Below the instructional label, the resultant slideshow will play when the user clicks on the "Start" button. This area is where the images from the selected folder will be displayed in a slideshow format.

Buttons: There are five buttons provided:

Load Image Folder: This button allows the user to load an image folder. When clicked, a file dialog box will open, enabling the user to select an image folder from their system.

Start slideshow: Clicking this button initiates the slideshow. The images from the selected folder will start displaying in the slideshow area above all buttons.

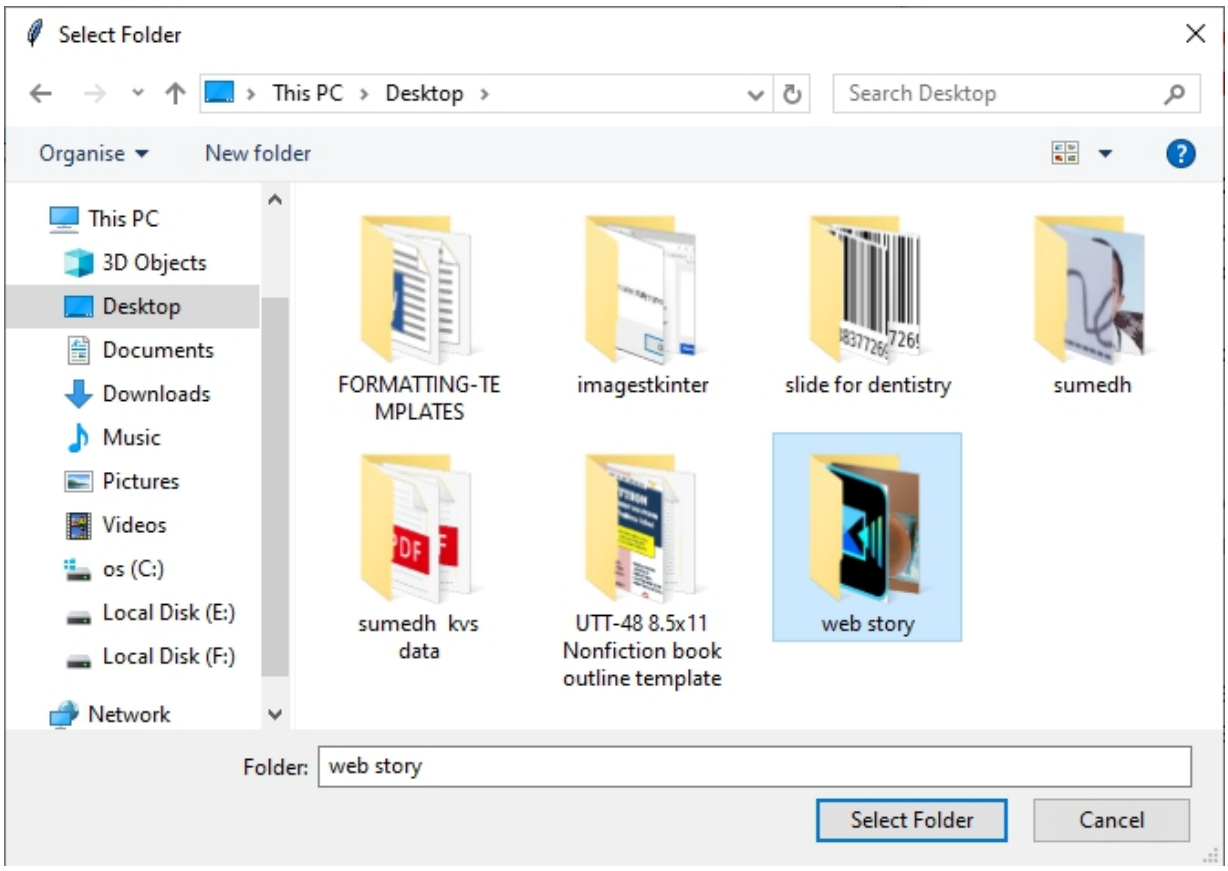
Stop slideshow: Clicking this button stops the slideshow altogether.

Pause slideshow: This button pauses the slideshow when clicked. The current image will remain displayed until the slideshow is resumed.

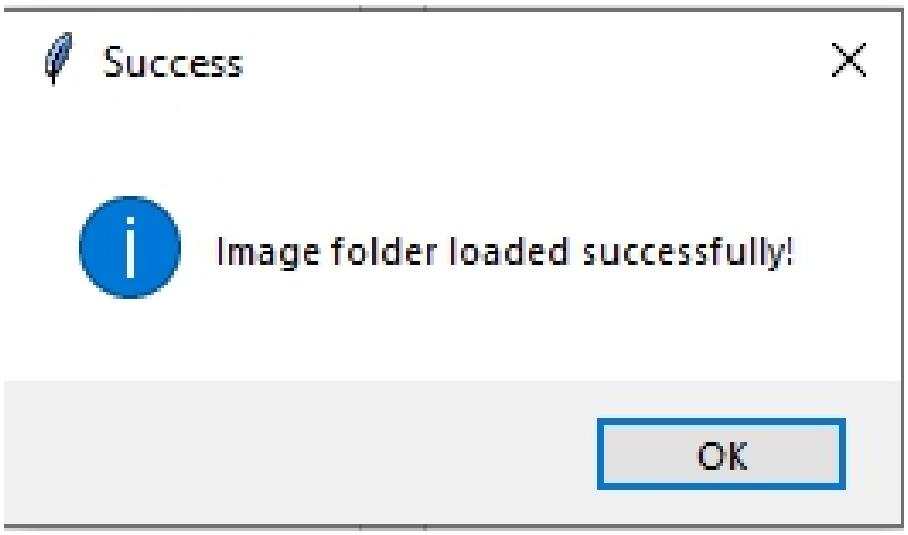
Resume slideshow: Clicking this button resumes the slideshow from where it was paused.

Messagebox: After uploading the image folder, a messagebox will appear, informing the user that the image folder has been successfully uploaded. This confirmation message provides feedback to the user about the completion of the upload process.

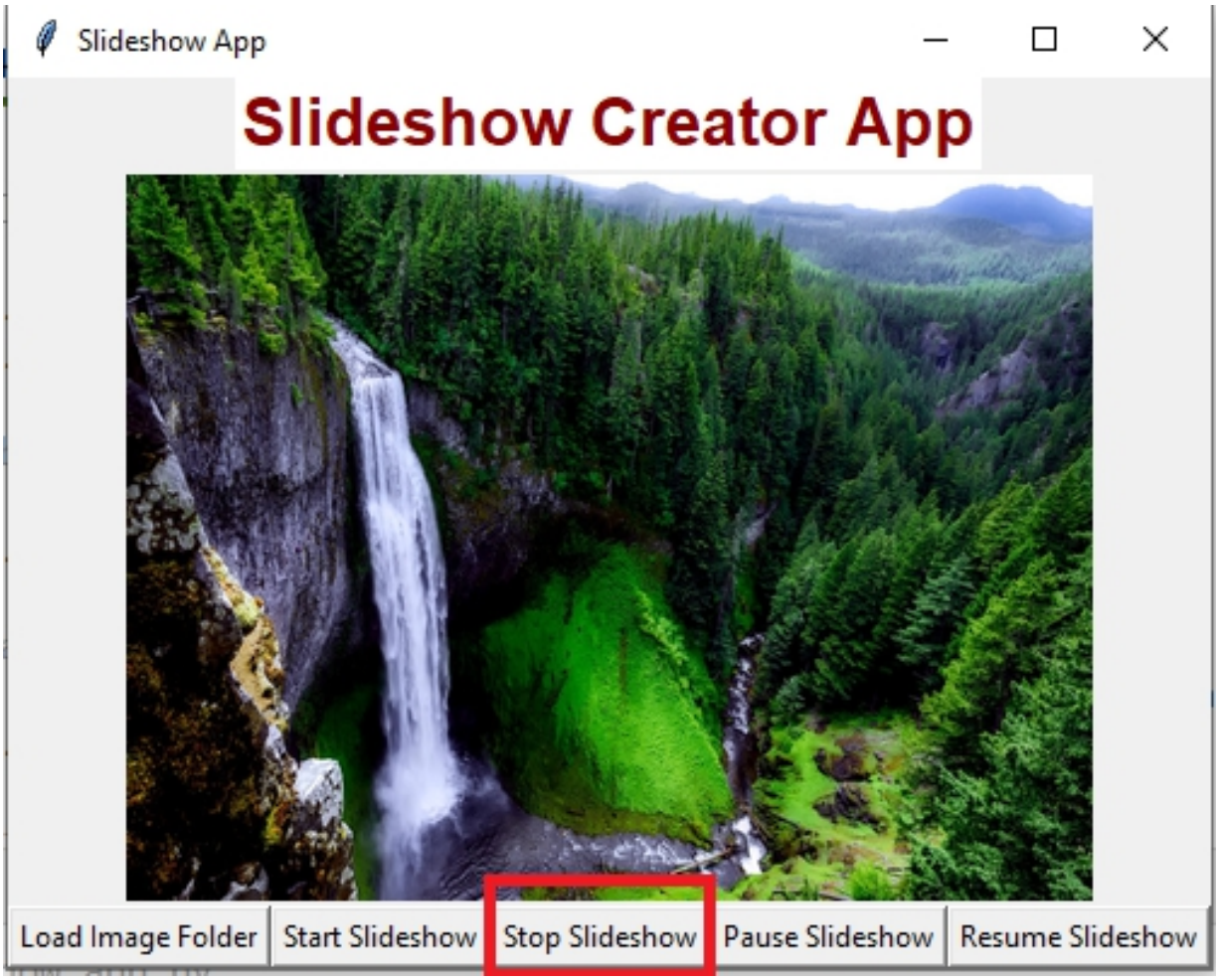
This is the main GUI of the Slideshow Creator App. When the user clicks on "Load Image Folder," a file dialogue box opens as shown below.



Here, the "Web Story" image folder is selected from the open file dialogue box, and it's successfully loaded into the application. A confirmation message box is then displayed to affirm the successful loading process.



When the user clicks the "Start Slideshow" button, the slideshow begins as shown above.



When the user clicks on the "Stop Slideshow" button, the slideshow is immediately halted.

Quadratic Equation Solver using the Factorisation Method

Project #19

Create a Python GUI for solving quadratic equations using Tkinter.

Introduction:

Welcome to the Quadratic Equation Solver using the Factorisation Method! This tool helps you solve quadratic equations effortlessly. Just type in the numbers for a, b, and c, and you'll get the answers you need.

As you enter the numbers, the equation appears right before your eyes. Once you're done, click the "Solve" button, and the app will find the solutions for you. If you need to start over, there's a handy "Clear All" button to erase everything.

It's like having a math wizard in your pocket, ready to tackle quadratic equations whenever you need it. Let's make math simpler and more fun together!

Lets do some mathematics to understand this concept

$x^2+x-20=0$

Lets talk about quadratic equation and its standard form. In this part, we do not go in deep. A quadratic equation is a second-degree polynomial equation in one variable, usually written in the form:

$ax^2 + bx + c = 0$

Here, a, b, and c are coefficients, and x is the variable. The highest power of the variable x is 2, which makes it a quadratic equation.

Quadratic equations can have zero, one, or two real solutions, depending on the values of the coefficients a, b, and c. These solutions represent the values of x where the equation is true.

Factorization of quadratic equations is the part of finding the roots of a quadratic equation. **Factoring quadratic equations** means converting the given quadratic expression into the product of two linear factors.

Lets understand this with example:

Question: Solve the quadratic equation $x^2 + x - 20 = 0$ by splitting the middle term.

Solution:

Given,

$x^2 + x -20 = 0$

Here, a = 1, b = 1, c = -20

ac = (1)(-20) = -20

Factors of -20: 1, 2, 4, 5, 10, 20

Let's identify two factors such that their sum is 1 and the product is -20.

Sum of two factors = 1 = -4+5

Product of these two factors = (5)(-4) = -20

Now, split the middle term.

x2 -4x + 5x -20 = 0

Take the common terms and simplify.

x(x - 4) + 5(x - 4) = 0

(x - 4)(x + 5) = 0

Thus, (x - 4) and (x + 5) are the factors of the given quadratic equation.

Solving these two linear factors, we get x = 4, -5 as roots.

Question: Solve the quadratic equation x2 + 7x + 10 = 0 by splitting the middle term.

Solution:

Given,

x2 + 7x + 10 = 0

Here, a = 1, b = 7, c = 10

ac = (1)(10) = 10

Factors of 10: 1, 2, 5, 10

Let’s identify two factors such that their sum is 7 and the product is 10.

Sum of two factors = 7 = 2 + 5

Product of these two factors = (2)(5) = 10

Now, split the middle term.

x2 + 2x + 5x + 10 = 0

Take the common terms and simplify.

x(x + 2) + 5(x + 2) = 0

(x + 5)(x + 2) = 0

Thus, (x + 2) and (x + 5) are the factors of the given quadratic equation.

Solving these two linear factors, we get x = -2, -5 as roots.

Code :

```
import tkinter as tk
from tkinter import messagebox
# Function to solve quadratic equation
def solve_quadratic(a, b, c):
    try:
        a = float(a)  # Convert coefficient a to float
        b = float(b)  # Convert coefficient b to float
        c = float(c)  # Convert coefficient c to float
        # Calculate the discriminant
        discriminant = b**2 - 4*a*c
        # Check discriminant to determine roots
        if discriminant < 0:
            return "No real roots"  # Return message if discriminant is negative
        elif discriminant == 0:
            x = -b / (2*a)  # Calculate single real root
            return f"One real root: x = {x}"  # Return message with single root
        else:
            # Calculate two real roots
```

```
            root1 = (-b + discriminant**0.5) / (2*a)
            root2 = (-b - discriminant**0.5) / (2*a)
            return f"Two real roots: x1 = {root1}, x2 = {root2}"  # Return message with two roots
    except ValueError:
        return "Invalid input. Please enter numeric values."  # Return message for invalid input

# Function to solve quadratic equation when 'Solve' button is clicked
def solve():
    result = solve_quadratic(entry_a.get(), entry_b.get(), entry_c.get())  # Solve quadratic equation
    messagebox.showinfo("Result", result)  # Display result in messagebox

# Function to clear all input fields when 'Clear All' button is clicked
def clear_all():
    entry_a.delete(0, tk.END)  # Clear coefficient a entry field
    entry_b.delete(0, tk.END)  # Clear coefficient b entry field
    entry_c.delete(0, tk.END)  # Clear coefficient c entry field
    equation_label.config(text=" ")  # Clear equation label text

# Create the main window
root = tk.Tk()
root.title("Quadratic Equation Solver")
```

 # Create a label for the title of the application

title_label = tk.Label(root, text="Quadratic Equation Solver using Factorisation method")

 # Place the title label in the root window, spanning across two columns, with padding

title_label.grid(row=0, column=0, columnspan=2, padx=5, pady=5)

 # Configure the appearance of the title label:

#set text color to Dark Blue,

#background color to white, and use a bold font

title_label.config(fg="Dark Blue", bg="white", font=("Helvetica", 15, "bold"))

 # instruction label for quadratic equation

note_label = tk.Label(root, text="Please note : Quadratic Equation form : ax^2 + bx + c = 0")

note_label.grid(row=1, column=0, columnspan=2, padx=5, pady=5)

note_label.config(fg="green", font=("helvetica",12,"bold"))

 # Create labels and entry widgets for coefficients a

label_a = tk.Label(root, text="Enter coefficient a:")

label_a.grid(row=2, column=0, padx=5, pady=5)

label_a.config(font=("Arial",10, "bold"))

entry_a = tk.Entry(root)

entry_a.grid(row=2, column=1, padx=5, pady=5)

 # Create labels and entry widgets for coefficients b

```
label_b = tk.Label(root, text="Enter coefficient b:")
label_b.grid(row=3, column=0, padx=5, pady=5)
label_b.config(font=("Arial",10, "bold"))
entry_b = tk.Entry(root)
entry_b.grid(row=3, column=1, padx=5, pady=5)
# Create labels and entry widgets for coefficients c
label_c = tk.Label(root, text="Enter coefficient c:")
label_c.grid(row=4, column=0, padx=5, pady=5)
label_c.config(font=("Arial",10, "bold"))
entry_c = tk.Entry(root)
entry_c.grid(row=4, column=1, padx=5, pady=5)
# Create a label to display the quadratic equation
equation_label = tk.Label(root, text=" ")
equation_label.grid(row=5, column=0, columnspan=2)
equation_label.config(fg="purple",font=("Arial",10, "bold"))
# Define a function to update the equation label with the quadratic equation
def update_equation():
    # Retrieve the coefficients 'a', 'b', and 'c' from the entry fields
    a = entry_a.get()
    b = entry_b.get()
    c = entry_c.get()
```

 # Update the equation label text to display the quadratic equation

 # based on the coefficients entered by the user

equation_label.config(text=f"Quadratic equation: {a}x^2 + {b}x + {c} = 0")

 # Bind the update_equation function to the FocusOut event

of the entry fields for coefficients 'a', 'b', and 'c'

entry_a.bind("<FocusOut>", lambda event: update_equation())

entry_b.bind("<FocusOut>", lambda event: update_equation())

entry_c.bind("<FocusOut>", lambda event: update_equation())

 # Create solve button

solve_button = tk.Button(root, text="Solve", command=solve)

solve_button.grid(row=6, column=0, columnspan=2, padx=5, pady=5)

solve_button.config(width=10, height=2, font=("helvetica", 10, "bold"))

 # Create solve button

clear_button = tk.Button(root, text="Clear All", command=clear_all)

clear_button.grid(row=6, column=1, padx=5, pady=5)

clear_button.config(width=10, height=2, font=("helvetica", 10, "bold"))

 # Run the main event loop

root.mainloop()

Output:

After executing the provided code, you will see a graphical user interface (GUI) window for the Quadratic Equation Solver using the Factorisation Method. Here's what you will see:

Title: At the center of the GUI window, there is a title label that reads "Quadratic Equation Solver using Factorisation Method."

Coefficient Entry Section: Below the title, there are three labels and entry widgets for entering the coefficients a, b, and c of the quadratic equation. Users can type in numeric values for these coefficients.

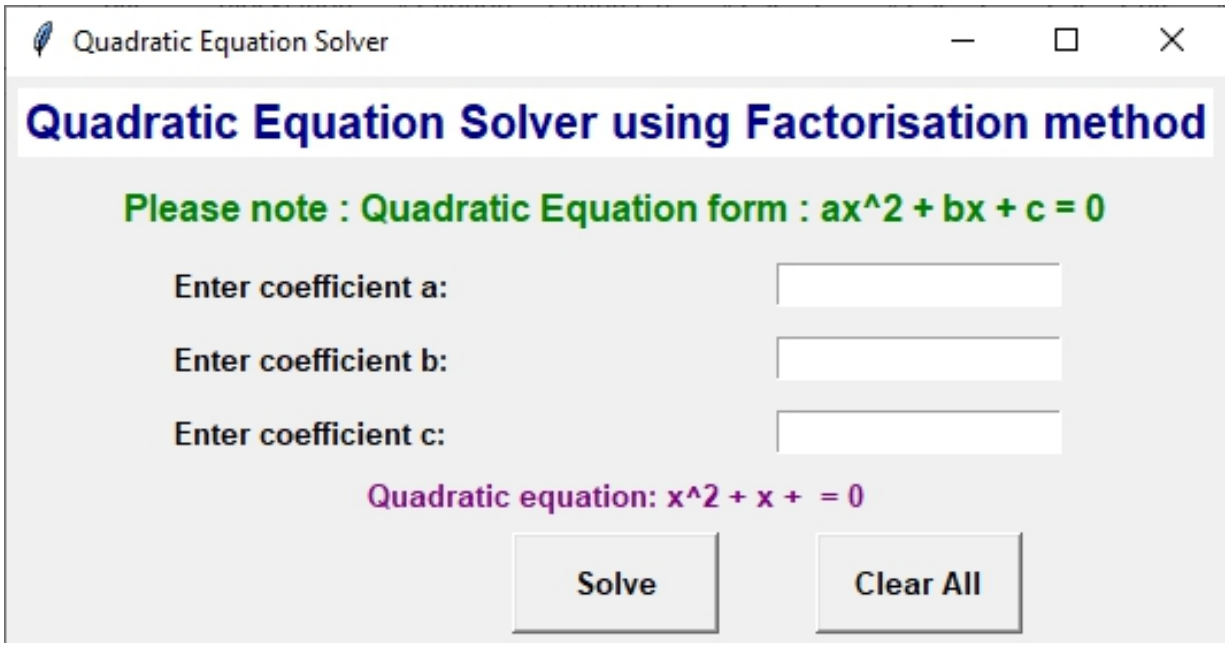
Quadratic Equation Form: Under the coefficient entry section, there is a label where the quadratic equation form is automatically updated as users type in the coefficients a, b, and c. This label displays the quadratic equation in the form ax^2+bx+c

Buttons: Below the quadratic equation form, there are two buttons:

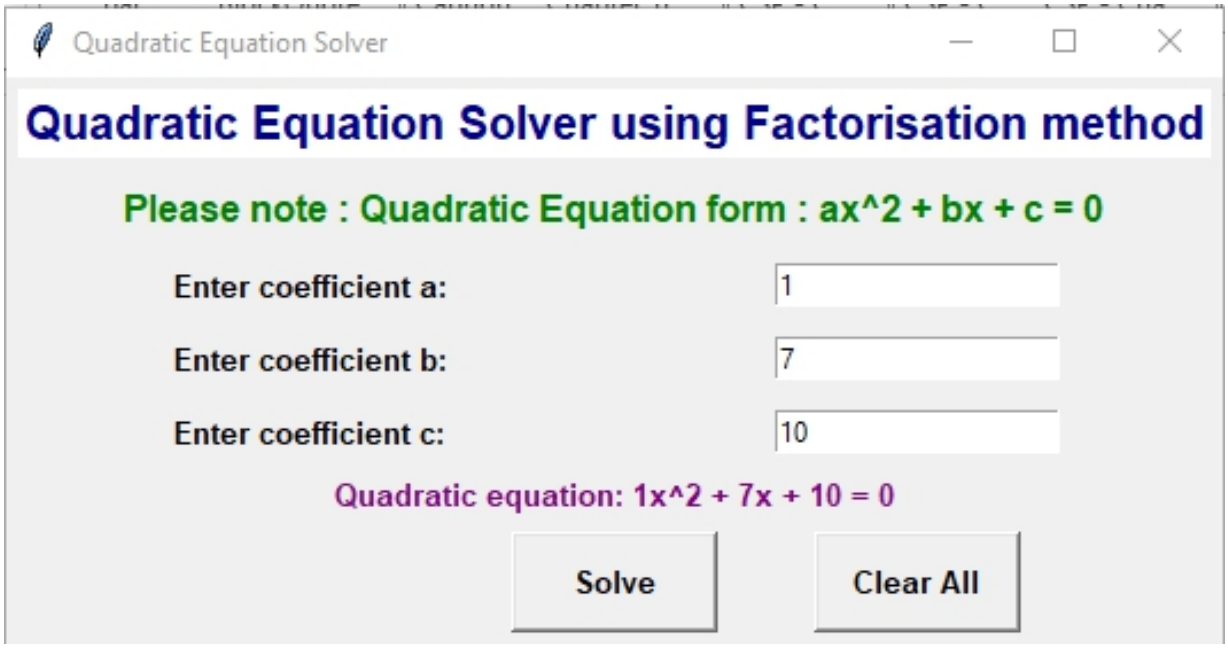
Solve: This button allows users to solve the quadratic equation based on the entered coefficients.

Clear All: This button clears all the entry widgets, allowing users to reset the coefficients and start over.

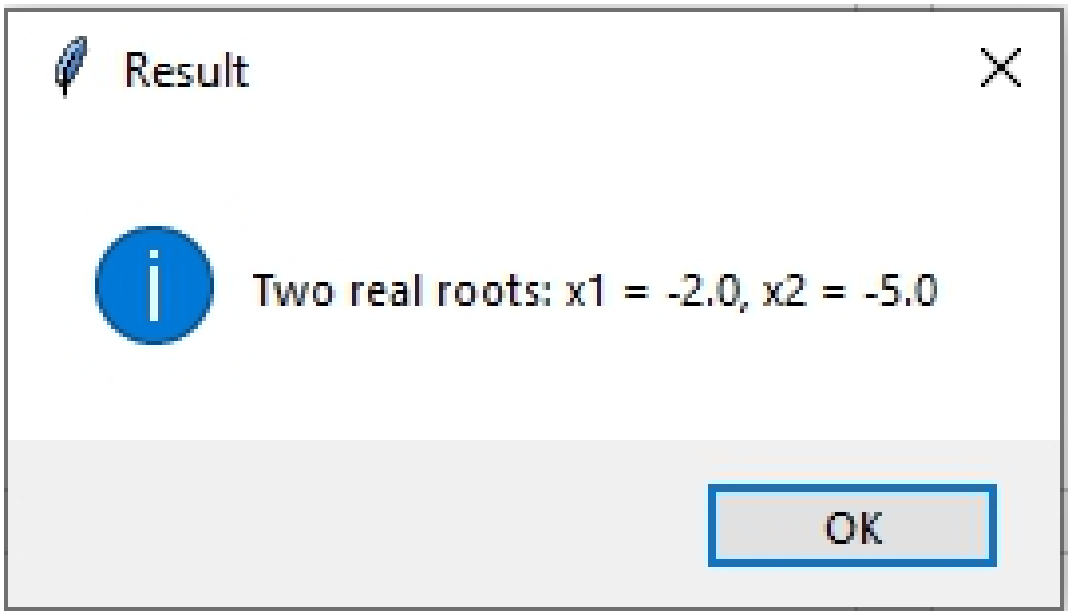
Result Display: After clicking the "Solve" button, a messagebox will appear displaying the roots of the quadratic equation if they exist. If the equation has no real roots or if the coefficients are invalid, appropriate error messages will be displayed in the messagebox.



This is the main GUI of the application, where users can find an instructional label displaying the correct form of a quadratic equation.



Here, the user enters the values of coefficients a, b, and c as 1, 7, and 10, respectively, as shown above, and the quadratic equation is automatically updated under the entry fields.



When the user clicks on the "Solve" button, the message above is displayed along with the result.

Events and Holiday Calender

Project #20

Design python GUI application for Events and Holidays Calender using tkinter

Introduction:

Introducing our last project of this book: the Events and Holidays Calendar. With this app, you can easily add your events and holidays to the calendar. When you click on any date, it automatically shows the event name. Plus, you can see all your events and holidays for the month in one place by clicking the "Show Events" or "Show Holidays" buttons. It's a simple way to keep track of your schedule and never miss an important day!

The aim of this project is to help you keep track of your schedule and never miss an important day. Plus, the app saves your events and holidays even after you close it, thanks to the shelve module. This means you can always access your data whenever you reopen the app.You can use this app to manage personal events, keep track of holidays, plan family activities, or even coordinate work schedules, making it a versatile tool for organizing your life.

Requirement :

Shelve module :

We will use **shelve** module to store events and holidays detail for specific dates in calender. Basically shelve module store data in a file on the local file system, typically store file in the current working directory of your python script unless you specify a different location. By default, it creates a file with the extension .db. It does not create traditional database like SQL or MySQL. Instaed it creates a

persistents storage file that acts like a dictionary, where Python objects are stored using keys for retrieval. In other words, we can simply store and retrieve Python objects persistently without the need for setting up and managing a database system.

To use **shelve** module in our project, We don’t need to install the shelve module separately using pip because its part of the Python standard library. It is always available in Python environment without the need of additional installations. We can simply import it using :

import shelve

tkcalendar module :

We will use tkcalendar module to create user friendly calendar interface

Code :

```
import tkinter as tk

from tkinter import ttk

from tkcalendar import Calendar

import shelve

# Load events and holidays from a shelve file

# Open the shelve file named 'calendar_data'

with shelve.open('calendar_data') as db:

    # Retrieve 'events' from the shelve file,

    # or an empty dictionary if not found

    events = db.get('events', {})

    # Retrieve 'holidays' from the shelve file,
```

```
    # or an empty dictionary if not found
    holidays = db.get('holidays', {})
# Function to add events to a particular date
def add_event():
    # Get the date entered by the user
    date = date_entry.get()
    # Get the event description entered by the user
    event = event_entry.get()
    # Add the event to the dictionary for the given date
    events.setdefault(date, []).append(event)
    # Update the display to show the newly added event
    update_event_label()
    # Save the updated events dictionary to the shelve file
    save_data()
    # Clear date and event entry fields
    date_entry.delete(0, tk.END)  # Clear the date entry field
    event_entry.delete(0, tk.END)  # Clear the event entry field
# Function to add holiday to a particular date
def add_holiday():
    # Get the date entered by the user
    date = date_entry.get()
```

```
    # Get the holiday description entered by the user
    holiday = holiday_entry.get()
    # Add the holiday to the dictionary for the given date
    holidays.setdefault(date, []).append(holiday)
    # Update the display to show the newly added holiday
    update_holiday_label()
    # Save the updated holidays dictionary to the shelve file
    save_data()
    # Clear date and holiday entry fields
    date_entry.delete(0, tk.END)  # Clear the date entry field
    holiday_entry.delete(0, tk.END)  # Clear the holiday entry field
# Function to save events and holidays to shelve file
def save_data():
    # Open the shelve file named 'calendar_data'
    with shelve.open('calendar_data') as db:
        # Save the events dictionary to the shelve file
        db['events'] = events
        # Save the holidays dictionary to the shelve file
        db['holidays'] = holidays
# Function to update the event label with events for the selected date
```

```
def update_event_label():
    # Get the selected date
    date = selected_date_label.cget("text")
    # Get the list of events for the selected date,
    # or an empty list if none
    event_list = events.get(date, [])
    # Update the event label to display the events
    event_label.config(text="\n".join(event_list))

# Function to update the holiday label with holidays for the selected date
def update_holiday_label():
    # Get the selected date
    date = selected_date_label.cget("text")
    # Get the list of holidays for the selected date,
    # or an empty list if none
    holiday_list = holidays.get(date, [])
    # Update the holiday label to display the holidays
    holiday_label.config(text="\n".join(holiday_list))

# Function to update the selected date label and the date entry field
def update_selected_date(date):
    # Update the selected date label to display the selected date
```

```
    selected_date_label.config(text=date)
    # Clear the date entry field
    date_entry.delete(0, tk.END)
    # Insert the selected date into the date entry field
    date_entry.insert(0, date)
     # Update the event label to display events for the selected date
    update_event_label()
    # Update the holiday label to display holidays for the selected date
    update_holiday_label()
# Function to show all events
def show_all_events():
    # Initialize an empty list to store all events
    event_list = []
    # Iterate over all dates and their corresponding events
    for date, events_list in events.items():
        for event in events_list:  # Iterate over all events for each date
            event_list.append(f"{date}: {event}")  # Add the event to the list
    # Update the label to display all events
    all_events_label.config(text="\n".join(event_list))
```

```
# Function to show all holidays
def show_all_holidays():
    holiday_list = []  # Initialize an empty list to store all holidays
    # Iterate over all dates and their corresponding holidays
    for date, holidays_list in holidays.items():
        # Iterate over all holidays for each date
        for holiday in holidays_list:
            # Add the holiday to the list
            holiday_list.append(f"{date}: {holiday}")
    # Update the label to display all holidays
    all_holidays_label.config(text="\n".join(holiday_list))
# Create main window
root = tk.Tk()  # Create the main window
root.title("Event Calendar") # Set the title of the window
root.geometry("400x300")  # Set the dimensions of the window
# Project title Label
title_label = tk.Label(root, text="Events and Holidays Calendar")  # Create a label for the project title
title_label.grid(row=0, column=0, padx=10, pady=10, columnspan=2) # Place the label in the window
title_label.config(fg="yellow", bg="blue", font=("Calibri", 30, "bold"))  # Configure the appearance of the label
```

 # Calendar widget

cal = Calendar(root, selectmode="day", year=2024, month=4) **# Create a calendar widget**

cal.grid(row=1, column=0, padx=10, pady=10, columnspan=2) **# Place the calendar in the window**

 # Label to display selected date

selected_date_label = ttk.Label(root, text="") **# Create a label to display the selected date**

selected_date_label.grid(row=2, column=0, padx=10, pady=5) **# Place the label in the window**

selected_date_label.config(font=("TkDefaultFont", 10, "bold")) **# Configure the appearance of the labe**l

 # Entry widget to enter date

date_label = ttk.Label(root, text="Enter Date (YYYY-MM-DD):") **# Create a label for the date entry**

date_label.grid(row=3, column=0, padx=10, pady=5) **# Place the label in the window**

date_label.config(font=("TkDefaultFont", 10, "bold")) **# Configure the appearance of the label**

date_entry = ttk.Entry(root, width=20) **# Create an entry widget for the date**

date_entry.grid(row=3, column=1, padx=10, pady=5) **# Place the entry widget in the window**

 # Entry widget to add events

event_label = ttk.Label(root, text="Enter Event:") **# Create a label for the event entry**

event_label.grid(row=4, column=0, padx=10, pady=5) # **Place the label in the window**

event_label.config(font=("TkDefaultFont", 10, "bold")) # **Configure the appearance of the label**

event_entry = ttk.Entry(root, width=20) # **Create an entry widget for the event**

event_entry.grid(row=4, column=1, padx=10, pady=5) # **Place the entry widget in the window**

Entry widget to add holidays

Create a label for the holiday entry

holiday_label = ttk.Label(root, text="Enter Holiday:") holiday_label.grid(row=5, column=0, padx=10, pady=5) # **Place the label in the window**

holiday_label.config(font=("TkDefaultFont", 10, "bold")) # Configure the appearance of the label

holiday_entry = ttk.Entry(root, width=20) # Create an entry widget for the holiday

holiday_entry.grid(row=5, column=1, padx=10, pady=5) # **Place the entry widget in the window**

Button to add events

add_event_button = ttk.Button(root, text="Add Event", command=add_event) # **Create a button to add events**

add_event_button.grid(row=6, column=0, padx=10, pady=5) # **Place the button in the window**

Button to add holidays

add_holiday_button = ttk.Button(root, text="Add Holiday", command=add_holiday) # **Create a button to add holidays**

add_holiday_button.grid(row=6, column=1, padx=10, pady=5) # **Place the button in the window**

Label to display events for the selected date

event_display_label = ttk.Label(root, text="Events for Selected Date:") # **Create a label for events**

event_display_label.grid(row=8, column=0, padx=10, pady=5) # **Place the label in the window**

event_label = ttk.Label(root, text="") # **Create a label to display events**

event_label.grid(row=9, column=0, padx=10, pady=5) # **Place the label in the window**

event_label.config(font=("TkDefaultFont", 10, "bold")) # **Configure the appearance of the label**

Label to display holidays for the selected date

holiday_display_label = ttk.Label(root, text="Holidays for Selected Date:") # **Create a label for holidays**

holiday_display_label.grid(row=8, column=1, padx=10, pady=5) # **Place the label in the window**

holiday_label = ttk.Label(root, text="") # **Create a label to display holidays**

holiday_label.grid(row=9, column=1, padx=10, pady=5) # **Place the label in the window**

holiday_label.config(font=("TkDefaultFont", 10, "bold"))

Button to show all events

show_all_events_button = ttk.Button(root, text="Show Events", command=show_all_events) # Create a button to show all events

show_all_events_button.grid(row=11, column=0, padx=10, pady=5) **# Place the button in the window**

 # Button to show all holidays

show_all_holidays_button = ttk.Button(root, text="Show Holidays", command=show_all_holidays) **# Create a button to show all holidays**

show_all_holidays_button.grid(row=11, column=1, padx=10, pady=5) **# Place the button in the window**

 # Label to display all events

all_events_label = ttk.Label(root, text="") **# Create a label to display all events**

all_events_label.grid(row=12, column=0, padx=10, pady=5) **# Place the label in the window**

all_events_label.config(font=("TkDefaultFont", 10, "bold")) **# Configure the appearance of the label**

 # Label to display all holidays

all_holidays_label = ttk.Label(root, text="") # Create a label to display all holidays

all_holidays_label.grid(row=12, column=1, padx=10, pady=5) # Place the label in the window

all_holidays_label.config(font=("TkDefaultFont", 10, "bold")) # Configure the appearance of the label

 # Bind the <<CalendarSelected>> event to update the selected date label and date entry field

```
cal.bind("<<CalendarSelected>>", lambda event:
update_selected_date(cal.get_date()))  # Bind the calendar event to update the selected date

root.mainloop()  # Start the main event loop
```

Output :

After running the code, you will see a GUI window with the following elements:

Title Label: The title "Events and Holidays Calendar" will be displayed at the top of the window.

Calendar Widget: Below the title, you'll find a calendar where you can select dates.

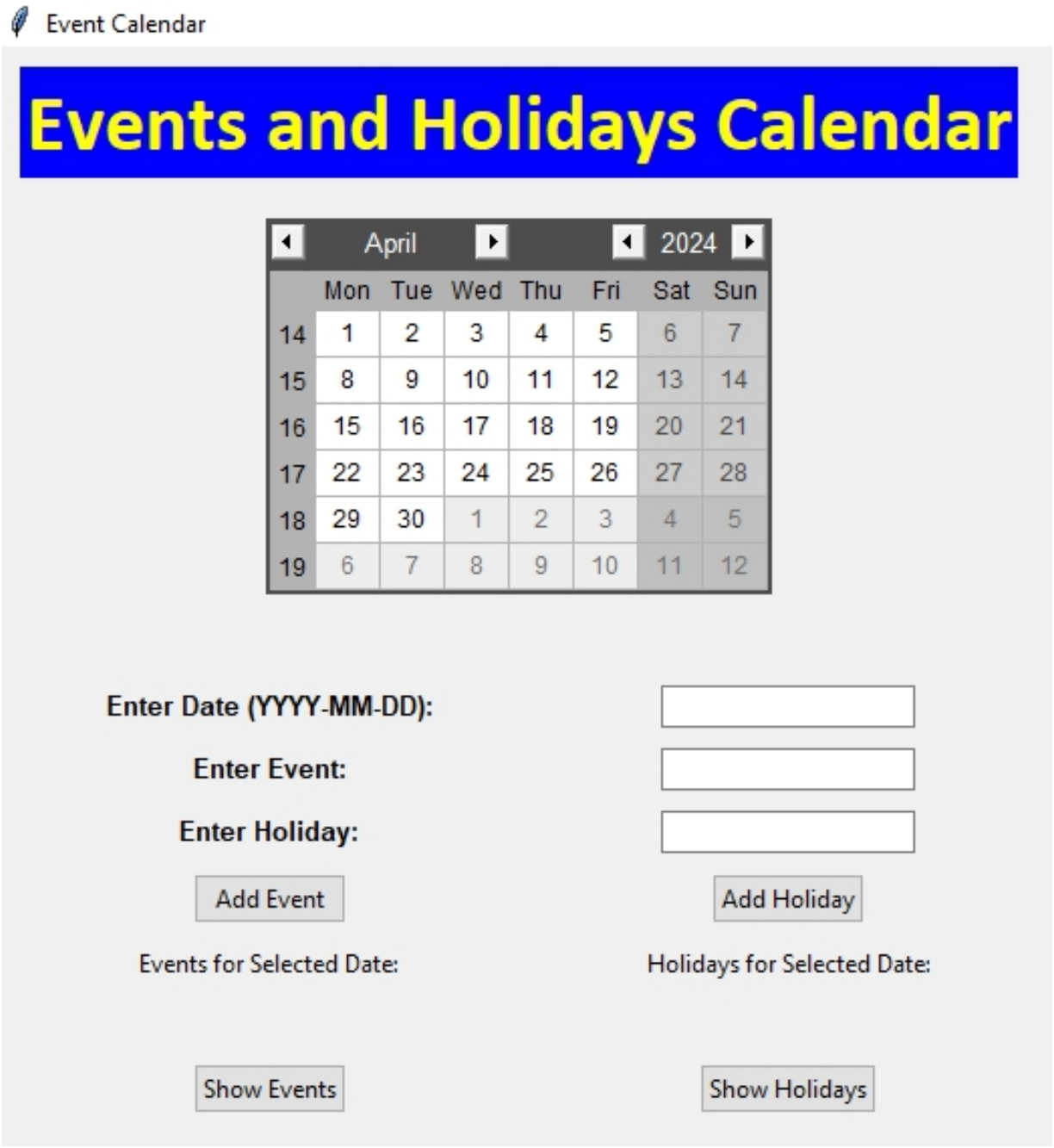
Entry Widgets: Under the calendar, there are entry fields where you can input the date, event name, and holiday name.

Add Event and Add Holiday Buttons: Two buttons labeled "Add Event" and "Add Holiday" allow you to add events and holidays, respectively, based on the date entered.

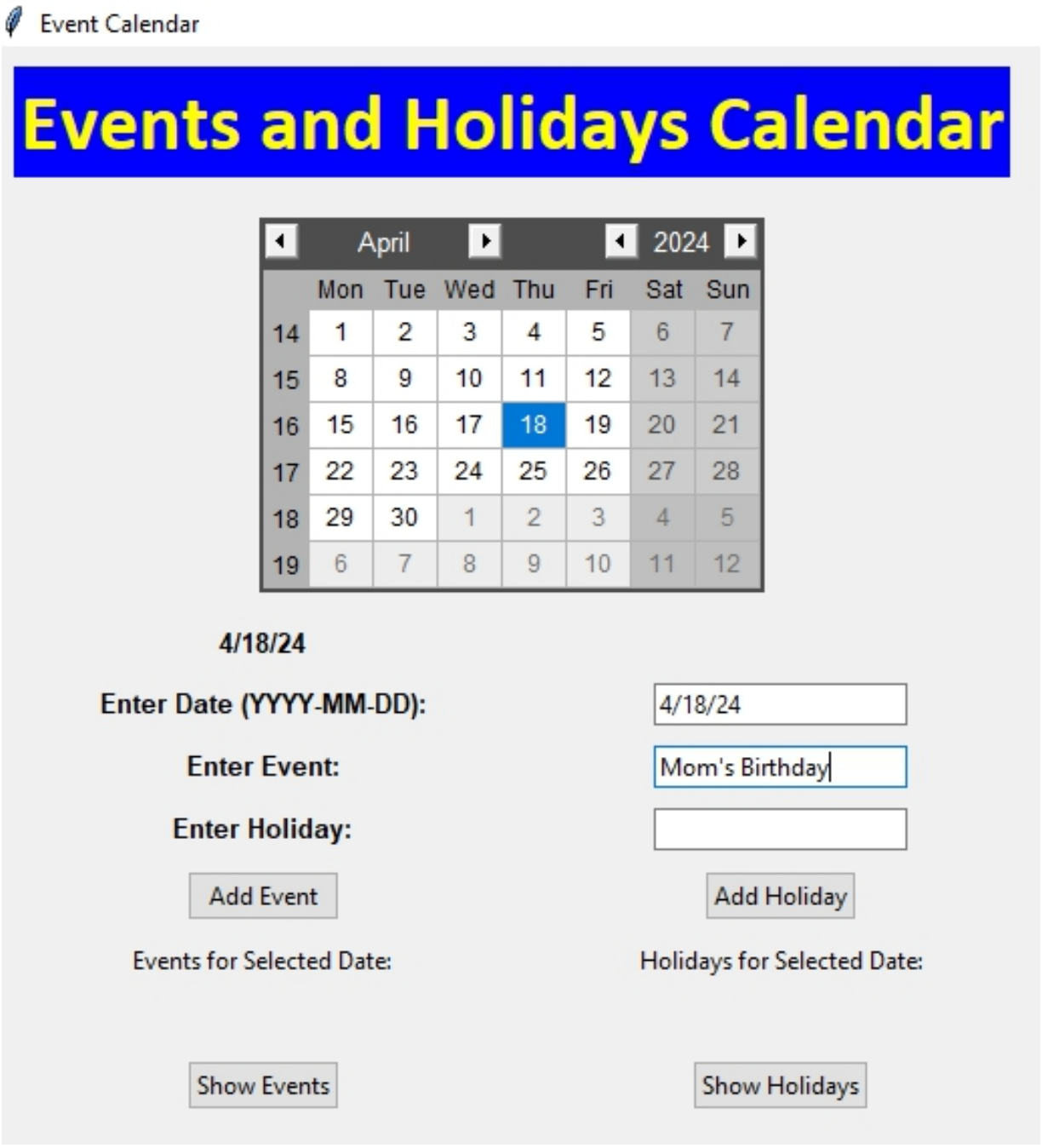
Labels for Events and Holidays: After adding events or holidays, labels titled "Events for Selected Date" and "Holidays for Selected Date" will display the respective events or holidays for the selected date.

Show All Events and Show All Holidays Buttons: Below the labels, there are buttons labeled "Show Events" and "Show Holidays". Clicking these buttons will display all events and holidays for the entire month.

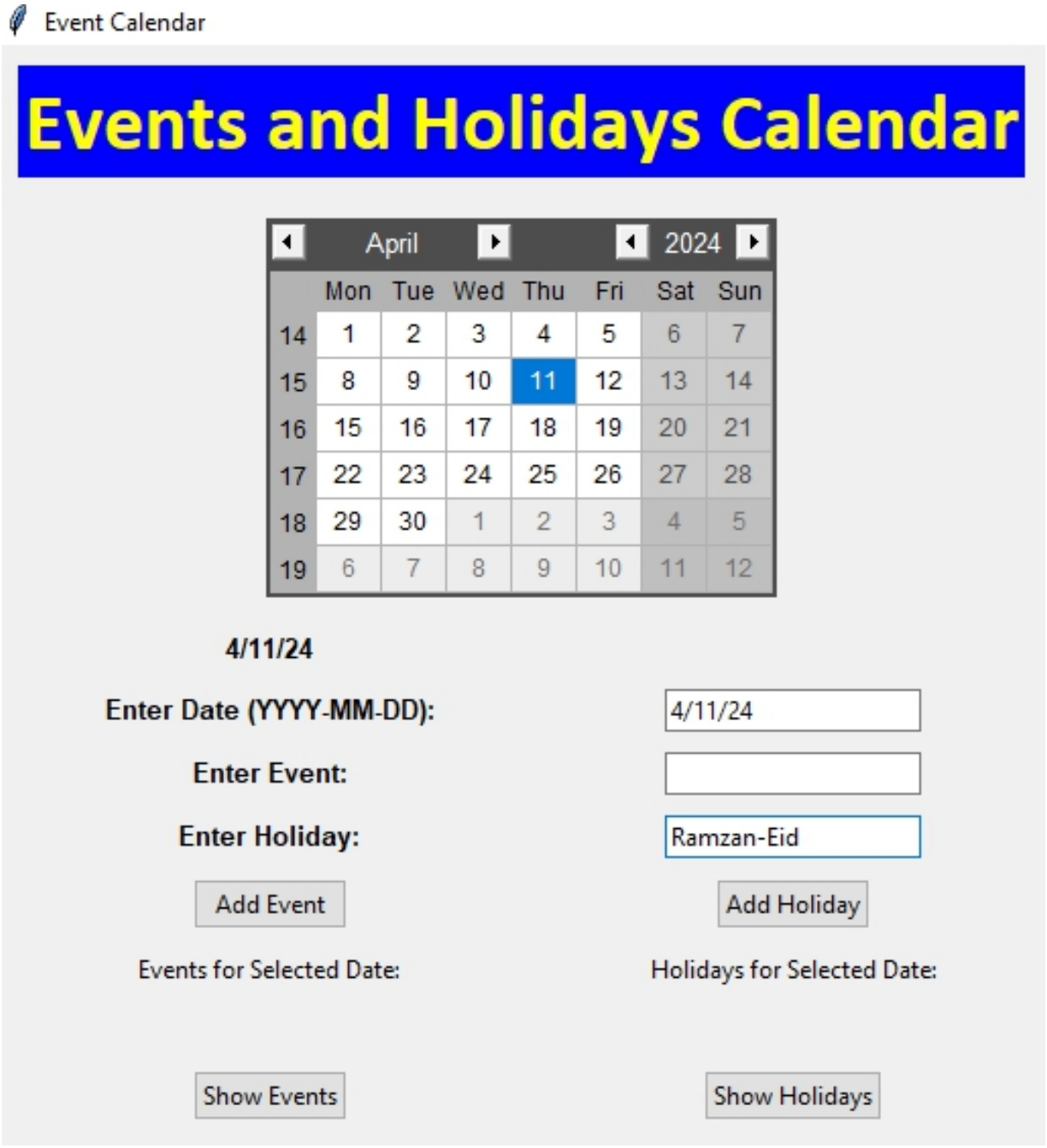
Labels for All Events and All Holidays: Under these buttons, there are labels that will show all events and holidays for the month once the corresponding button is clicked.



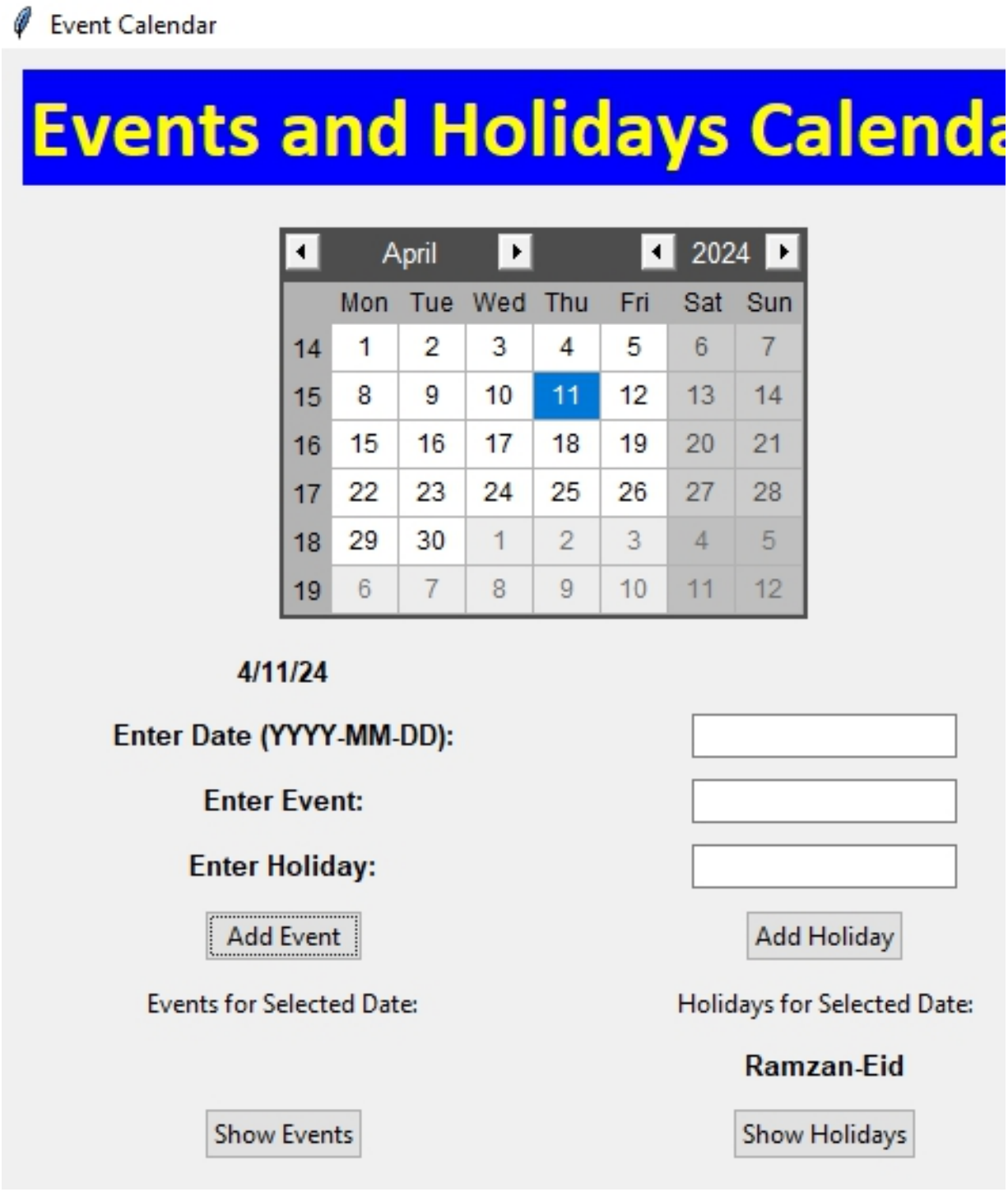
This is main window of GUI



When the user clicks on a date in the calendar, the selected date is automatically populated in the date entry field. For instance, if the user selects the date 18/4/2024, it will appear in the date entry field. The user can then add event names for the selected date. For example, if the user enters "Mom's birthday" as the event name and clicks on the "Add Event" button, it will be added to the selected date. The same process can be followed for adding holidays.

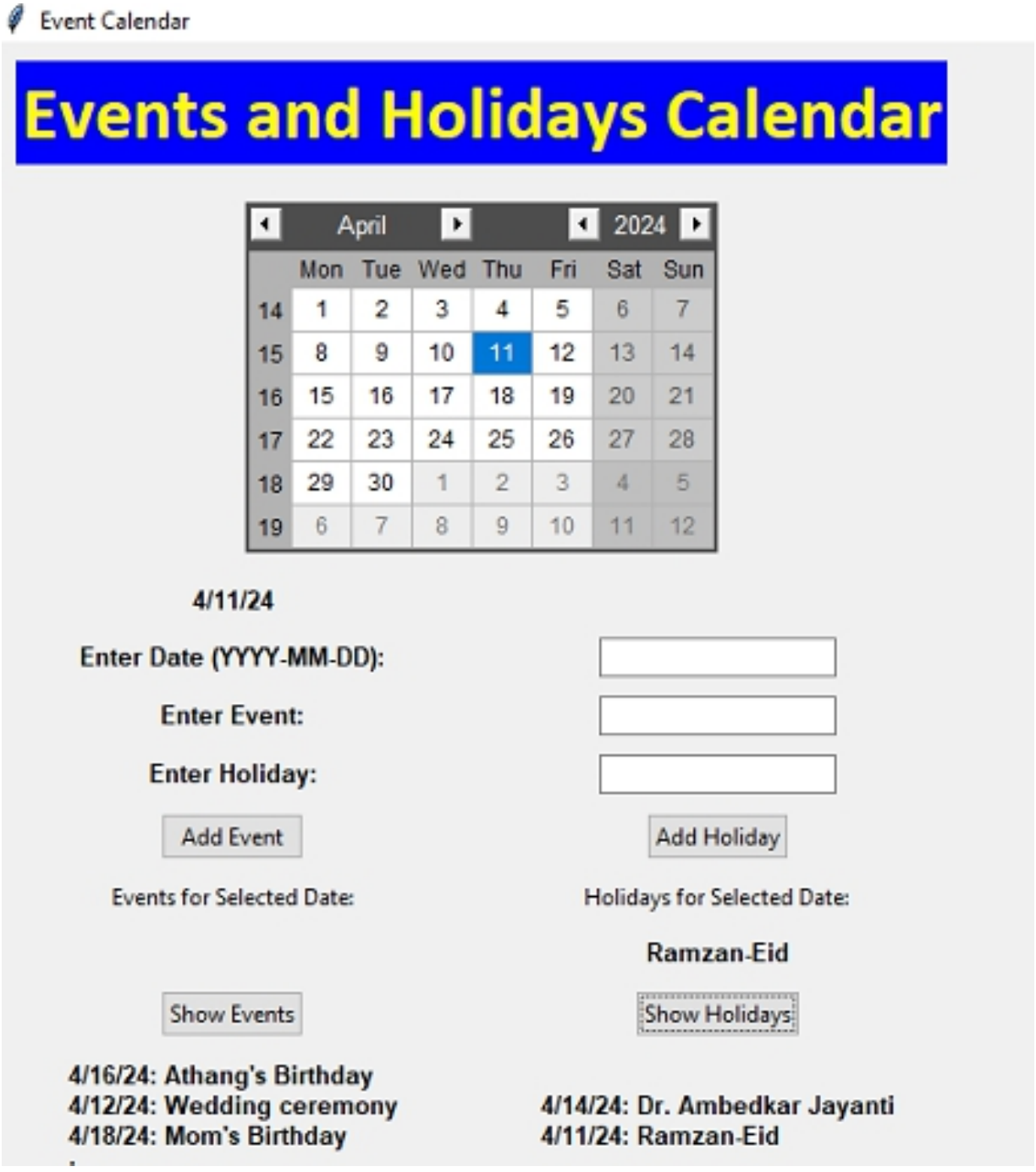


In the GUI displayed above, the date 11/4/2024 is currently selected, and the user has added the holiday name "Ramzan-Eid". Upon clicking the "Add Holiday" button, this holiday name is successfully added to the selected date, as illustrated.



When the user clicks on any date, the corresponding events or holiday names are displayed, as demonstrated above. For instance, we recently added the holiday on 11/4/2024. Therefore, when the

user clicks on this date, the holiday name "Ramzan-Eid" is displayed, as shown above.



When the user clicks on the "Show Events" button, all events for that month will be shown. In this instance, it displays all events for the month of April, and the same applies for holidays.

About the Author

Ms. Vaishali B. Bhagat received her BE in Information Technology and a Master's degree (M.Tech) in Computer Science from Nagpur University. She worked as an Assistant Professor in an engineering college for 10 years. She is also a passionate teacher and has taught Python programming to students of all ages. She authored books on Python programming titled “Learn Python with 200 Programs,” “Python Assignment Solution: 500 Problems Solved,” and "Python tkinter: 35 Mini Projects." The first book contains elementary programs as well as typical ones, while the second provides solutions for conceptual questions, knowledge-based questions, and short and long-length questions, explained in detail. The third book contains 35 mini Python projects that are very useful for starting coding in Python; they are well-written and correctly compiled projects. All three books are intended for novices and beginners interested in the Python programming language.

Author website : https:\\www.technocrash.cloud