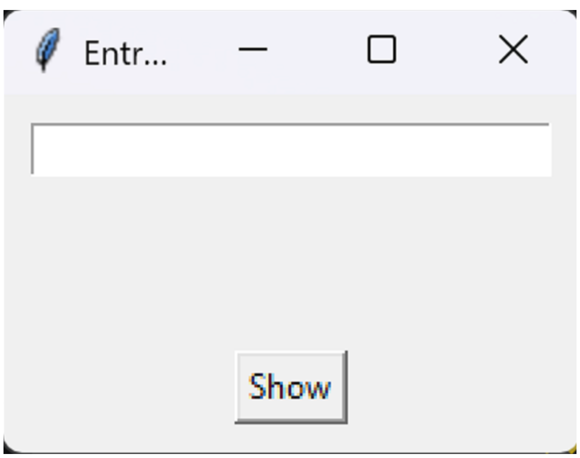
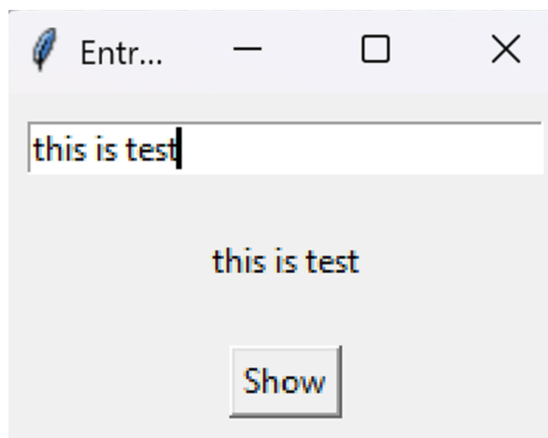
Q1: Build a Tkinter window with an Entry and a Button. When the button is clicked, show the entered text in a Label.

Solution:

```
import tkinter as tk

root = tk.Tk()
root.title("Entry to Label")

entry = tk.Entry(root, width=30)
entry.pack(padx=10, pady=10)

result_label = tk.Label(root, text="")
result_label.pack(padx=10, pady=10)


def show_text():
    result_label.config(text=entry.get())


btn = tk.Button(root, text="Show", command=show_text)
btn.pack(padx=10, pady=10)

root.mainloop()
```



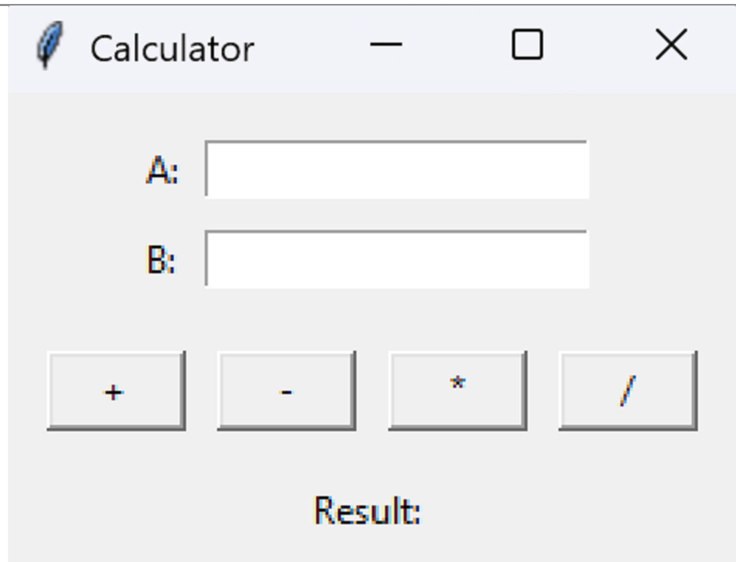


Q2: Create a calculator UI (two Entry boxes for numbers, buttons: + - * /, and a result Label). Add input validation (handle empty and non-numeric input).

Solution:

```
import tkinter as tk
from tkinter import messagebox

def get_numbers():
    a_text = entry_a.get().strip()
    b_text = entry_b.get().strip()

    if not a_text or not b_text:
        messagebox.showerror("Input Error", "Please enter both numbers.")
        return None
    try:
        a = float(a_text)
        b = float(b_text)
    except ValueError:
        messagebox.showerror("Input Error", "Please enter valid numbers.")
        return None

    return a, b

def calculate(op):
    nums = get_numbers()
    if nums is None:
        return

    a, b = nums

    if op == "+":
        res = a + b
    elif op == "-":
        res = a - b
    elif op == "*":
        res = a * b
    else:
        if b == 0:
            messagebox.showerror("Math Error", "Cannot divide by zero.")
            return
        res = a / b

    result_label.config(text=f"Result: {res}")
root = tk.Tk()
root.title("Calculator")

frm = tk.Frame(root)
frm.pack(padx=10, pady=10)

tk.Label(frm, text="A:").grid(row=0, column=0, sticky="w")
entry_a = tk.Entry(frm, width=20)
entry_a.grid(row=0, column=1, padx=5, pady=5)

tk.Label(frm, text="B:").grid(row=1, column=0, sticky="w")
entry_b = tk.Entry(frm, width=20)
entry_b.grid(row=1, column=1, padx=5, pady=5)

btns = tk.Frame(root)
```

```
btns.pack(padx=10, pady=5)

for i, op in enumerate(["+", "-", "*", "/"]):
    tk.Button(btns, text=op, width=5, command=lambda o=op: calculate(o)).grid(row=0, column=i,
padx=5)

result_label = tk.Label(root, text="Result: ")
result_label.pack(padx=10, pady=10)

root.mainloop()
```

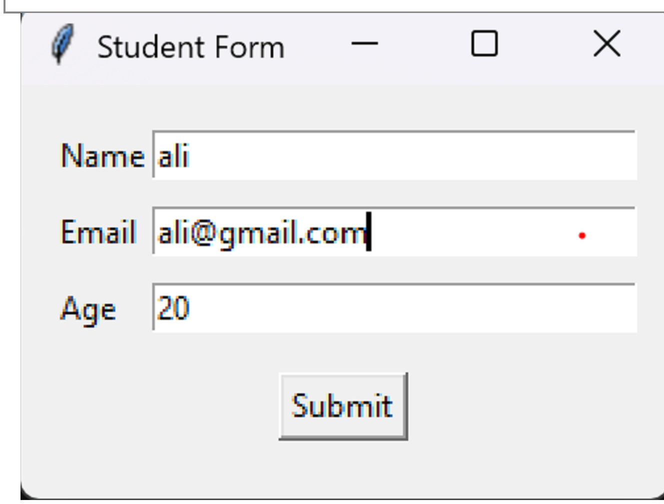


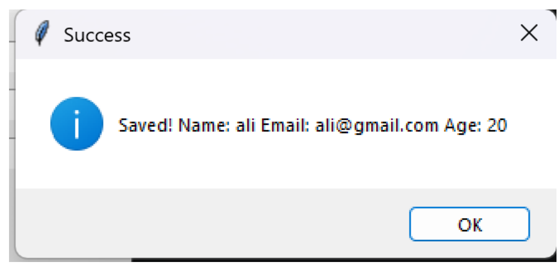
Q3: Build a form (Name, Email, Age) using grid. On Submit, validate fields and show errors using messagebox.

Solution:

```
import tkinter as tk
from tkinter import messagebox
def submit():
    name = entry_name.get().strip()
    email = entry_email.get().strip()
    age_text = entry_age.get().strip()

    if not name:
        messagebox.showerror("Validation", "Name is required.")
        return

    if "@" not in email or "." not in email:
        messagebox.showerror("Validation", "Email is not valid.")
        return

    try:
        age = int(age_text)
        if age <= 0:
            raise ValueError
    except ValueError:
        messagebox.showerror("Validation", "Age must be a positive integer.")
        return
```

```
    messagebox.showinfo("Success", f"Saved! Name: {name} Email: {email} Age: {age}")

root = tk.Tk()
root.title("Student Form")

frm = tk.Frame(root)
frm.pack(padx=12, pady=12)


tk.Label(frm, text="Name").grid(row=0, column=0, sticky="w", pady=4)
entry_name = tk.Entry(frm, width=30)
entry_name.grid(row=0, column=1, pady=4)

tk.Label(frm, text="Email").grid(row=1, column=0, sticky="w", pady=4)
entry_email = tk.Entry(frm, width=30)
entry_email.grid(row=1, column=1, pady=4)

tk.Label(frm, text="Age").grid(row=2, column=0, sticky="w", pady=4)
entry_age = tk.Entry(frm, width=30)
entry_age.grid(row=2, column=1, pady=4)


tk.Button(frm, text="Submit", command=submit).grid(row=3, column=0, columnspan=2, pady=10)

root.mainloop()
```



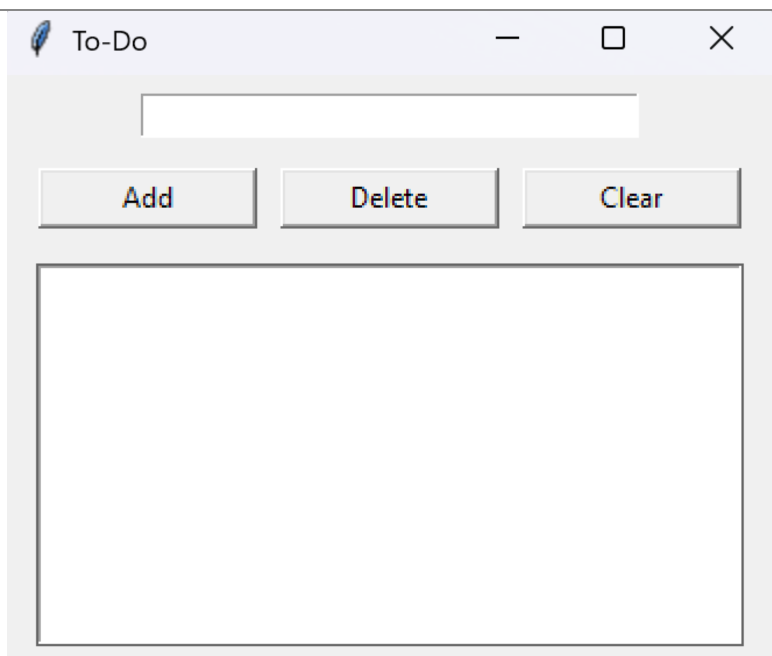


Q4: Create a To-Do app using a Listbox with buttons: Add, Delete selected, Clear all. Add items from an Entry.

Solution:

```
import tkinter as tk
from tkinter import messagebox


def add_item():
    text = entry.get().strip()
    if not text:
        messagebox.showwarning("To-Do", "Enter a task.")
        return
    listbox.insert(tk.END, text)
```

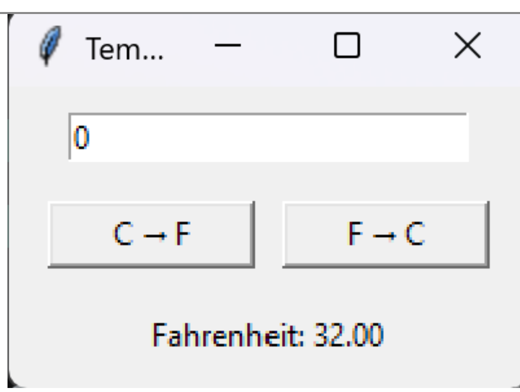
```
    entry.delete(0, tk.END)


def delete_selected():
    sel = listbox.curselection()
    if not sel:
        messagebox.showwarning("To-Do", "Select an item to delete.")
        return
    listbox.delete(sel[0])


def clear_all():
    listbox.delete(0, tk.END)


root = tk.Tk()
root.title("To-Do")

entry = tk.Entry(root, width=35)
entry.pack(padx=10, pady=8)

btns = tk.Frame(root)
btns.pack(padx=10, pady=5)


tk.Button(btns, text="Add", width=12, command=add_item).grid(row=0, column=0, padx=5)
tk.Button(btns, text="Delete", width=12, command=delete_selected).grid(row=0, column=1, padx=5)
tk.Button(btns, text="Clear", width=12, command=clear_all).grid(row=0, column=2, padx=5)

listbox = tk.Listbox(root, width=50, height=10)
listbox.pack(padx=10, pady=10)

root.mainloop()
```



Q5: Create a Celsius ↔ Fahrenheit converter. Use an Entry, two buttons (C→F and F→C), and a result Label. Validate input.

Solution:

```
import tkinter as tk
from tkinter import messagebox
def read_value():
    t = entry.get().strip()
    if not t:
        messagebox.showerror("Input", "Enter a value.")
        return None
    try:
        return float(t)
    except ValueError:
        messagebox.showerror("Input", "Enter a valid number.")
        return None
def c_to_f():
    c = read_value()
    if c is None:
        return
    f = c * 9 / 5 + 32
    result.config(text=f"Fahrenheit: {f:.2f}")
def f_to_c():
    f = read_value()
    if f is None:
        return
    c = (f - 32) * 5 / 9
    result.config(text=f"Celsius: {c:.2f}")
root = tk.Tk()
root.title("Temperature Converter")

entry = tk.Entry(root, width=25)
entry.pack(padx=10, pady=10)

btns = tk.Frame(root)
btns.pack(padx=10, pady=5)

tk.Button(btns, text="C → F", width=10, command=c_to_f).grid(row=0, column=0, padx=5)
tk.Button(btns, text="F → C", width=10, command=f_to_c).grid(row=0, column=1, padx=5)

result = tk.Label(root, text="")
result.pack(padx=10, pady=10)

root.mainloop()
```



Q6: Create a class Circle with radius. Add methods area(), circumference(), and scale(k) (multiply radius by k). Test it.

Solution:

```
import math
class Circle:
    def __init__(self, radius):
        self.radius = float(radius)
    def area(self):
        return math.pi * self.radius ** 2

    def circumference(self):
        return 2 * math.pi * self.radius

    def scale(self, k):
        self.radius *= float(k)
c = Circle(5)
print("radius=", c.radius)
print("area=", c.area())
print("circ=", c.circumference())

c.scale(2)
print("after scale radius=", c.radius)
print("area=", c.area())
```

Q7: Create a class Student and override __str__() to print: Name(ID) avg=... . Add method from_string("Ali,101,80,70,90") to create an object.

Solution:

```
class Student:
    def __init__(self, name, student_id, grades=None):
        self.name = name
        self.id = int(student_id)
        self.grades = grades if grades is not None else []
    def average(self):
        return sum(self.grades) / len(self.grades) if self.grades else 0.0
    def __str__(self):
        return f"{self.name}({self.id}) avg={self.average():.2f}"
    @classmethod
    def from_string(cls, text):
        parts = [p.strip() for p in text.split(",")]
        name = parts[0]
        student_id = int(parts[1])
        grades = [float(x) for x in parts[2:]]
        return cls(name, student_id, grades)
s = Student.from_string("Ali,101,80,70,90")
print(s)
```

Q8: Create a class InventoryItem with name, price, stock. Add methods sell(qty) (decrease stock if available) and restock(qty).

Solution:

```
class InventoryItem:
    def __init__(self, name, price, stock):
        self.name = name
        self.price = float(price)
        self.stock = int(stock)

    def sell(self, qty):
        qty = int(qty)
        if qty <= 0:
            return False
        if qty > self.stock:
            return False
        self.stock -= qty
        return True

    def restock(self, qty):
        qty = int(qty)
        if qty <= 0:
            return False
        self.stock += qty
        return True
item = InventoryItem("USB", 5.0, 10)
print(item.name, item.stock)
print("sell 3:", item.sell(3), "stock:", item.stock)
print("sell 20:", item.sell(20), "stock:", item.stock)
print("restock 5:", item.restock(5), "stock:", item.stock)
```

Q9: Create classes Point(x,y) and Line(p1,p2). In Line, add method length() (use math). Test with sample points.

Solution:

```
import math
class Point:
    def __init__(self, x, y):
        self.x = float(x)
        self.y = float(y)
class Line:
    def __init__(self, p1, p2):
        self.p1 = p1
        self.p2 = p2

    def length(self):
        dx = self.p2.x - self.p1.x
        dy = self.p2.y - self.p1.y
        return math.sqrt(dx * dx + dy * dy)
a = Point(0, 0)
b = Point(3, 4)
line = Line(a, b)
print("length=", line.length())
```

Q10:Create a class GradeBook that stores many students (name → list of grades) using a dictionary. Methods: add_student(name), add_grade(name, grade), student_avg(name), top_student().

Solution:

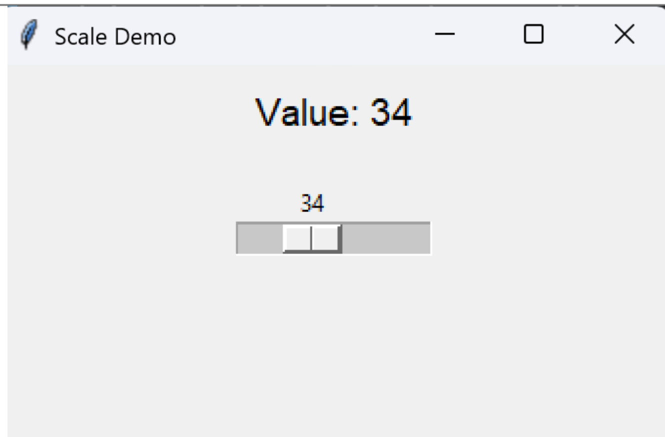
```
class GradeBook:
    def __init__(self):
        self.data = {}
    def add_student(self, name):
        if name not in self.data:
            self.data[name] = []

    def add_grade(self, name, grade):
        if name not in self.data:
            self.add_student(name)
        self.data[name].append(float(grade))
    def student_avg(self, name):
        grades = self.data.get(name, [])
        return sum(grades) / len(grades) if grades else 0.0

    def top_student(self):
        if not self.data:
            return None
        return max(self.data.keys(), key=lambda n: self.student_avg(n))
gb = GradeBook()
gb.add_grade("Ali", 80)
gb.add_grade("Ali", 70)
gb.add_grade("Sara", 90)
gb.add_grade("Sara", 95)
gb.add_grade("Omar", 40)

print("Ali avg=", gb.student_avg("Ali"))
print("Top student=", gb.top_student())
```

Q11: Build a Tkinter window with a Scale (0–100). When the slider moves, update a Label to show the current value.

Solution:

```
import tkinter as tk
def on_change(value):
    lbl.config(text=f"Value: {int(float(value))}")


root = tk.Tk()
root.title("Scale Demo")

lbl = tk.Label(root, text="Value: 0", font=("Arial", 14))
lbl.pack(padx=10, pady=10)

scale = tk.Scale(root, from_=0, to=100, orient="horizontal", command=on_change)
```

```
scale.pack(padx=10, pady=10)

root.mainloop()
```

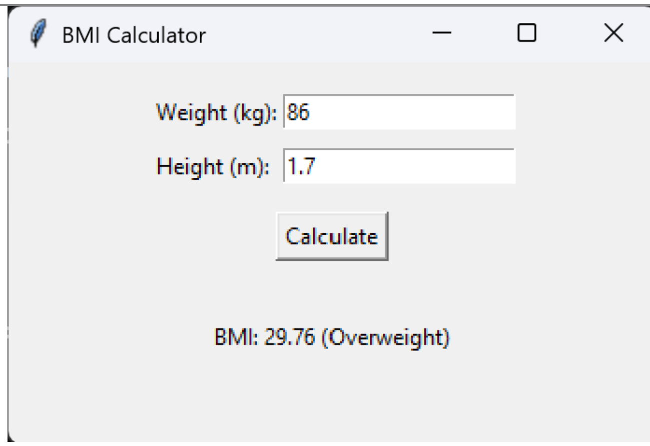


Q12: Create a BMI Calculator UI: Entry for weight (kg) and height (m), a Button to calculate, and a result Label. Validate input.

Solution:

```
import tkinter as tk
from tkinter import messagebox
def calculate():
    w_text = entry_w.get().strip()
    h_text = entry_h.get().strip()

    try:
        w = float(w_text)
        h = float(h_text)
        if w <= 0 or h <= 0:
            raise ValueError
    except ValueError:
        messagebox.showerror("Input", "Enter valid positive numbers for weight and height.")
        return

    bmi = w / (h * h)

    if bmi < 18.5:
        status = "Underweight"
    elif bmi < 25:
        status = "Normal"
    elif bmi < 30:
        status = "Overweight"
    else:
        status = "Obese"

    result.config(text=f"BMI: {bmi:.2f} ({status})")
root = tk.Tk()
root.title("BMI Calculator")

frm = tk.Frame(root)
frm.pack(padx=12, pady=12)

tk.Label(frm, text="Weight (kg):").grid(row=0, column=0, sticky="w", pady=4)
entry_w = tk.Entry(frm, width=20)
```

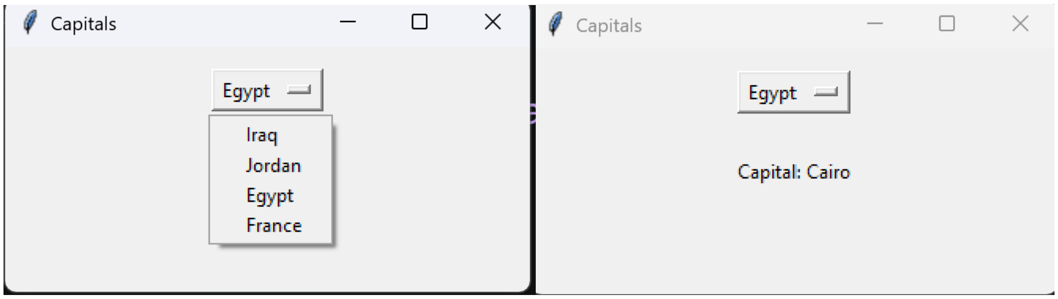
```
entry_w.grid(row=0, column=1, pady=4)

tk.Label(frm, text="Height (m):").grid(row=1, column=0, sticky="w", pady=4)
entry_h = tk.Entry(frm, width=20)
entry_h.grid(row=1, column=1, pady=4)

tk.Button(frm, text="Calculate", command=calculate).grid(row=2, column=0, columnspan=2, pady=10)

result = tk.Label(root, text="")
result.pack(padx=12, pady=8)

root.mainloop()
```



Q13: Create OptionMenu to choose a country, then show its capital in a Label (use a dictionary).

Solution:

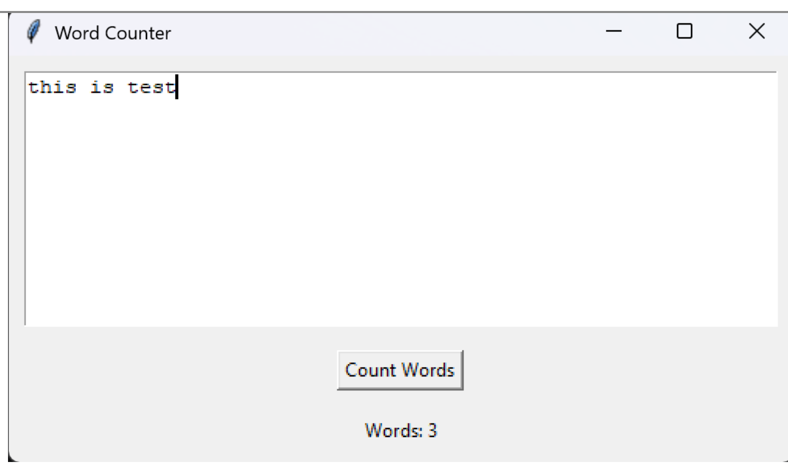
```
import tkinter as tk
capitals = {
    "Iraq": "Baghdad",
    "Jordan": "Amman",
    "Egypt": "Cairo",
    "France": "Paris",
}
def update_capital(*_):
    country = country_var.get()
    capital = capitals.get(country, "")
    lbl.config(text=f"Capital: {capital}")
root = tk.Tk()
root.title("Capitals")

country_var = tk.StringVar(value="Iraq")
menu = tk.OptionMenu(root, country_var, *capitals.keys())
menu.pack(padx=12, pady=12)

lbl = tk.Label(root, text="")
lbl.pack(padx=12, pady=12)

country_var.trace_add("write", update_capital)
update_capital()

root.mainloop()
```



Q14: Build an app with a Text widget and a Button "Count Words". Show the word count in a Label.

Solution:

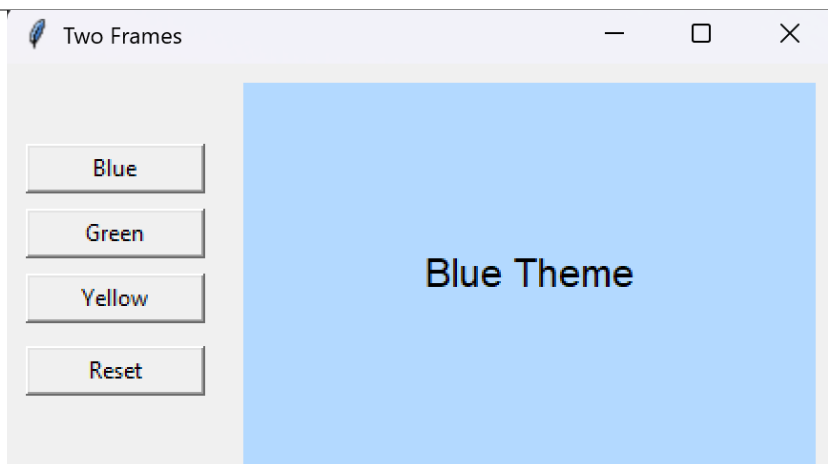
```
import tkinter as tk

def count_words():
    text = box.get("1.0", tk.END)
    words = [w for w in text.split() if w.strip()]
    lbl.config(text=f"Words: {len(words)}")
root = tk.Tk()
root.title("Word Counter")

box = tk.Text(root, width=60, height=10)
box.pack(padx=10, pady=10)

btn = tk.Button(root, text="Count Words", command=count_words)
btn.pack(padx=10, pady=5)

lbl = tk.Label(root, text="Words: 0")
lbl.pack(padx=10, pady=10)

root.mainloop()
```



Q15: Create an app with two frames: left has buttons, right is a preview area. Buttons change preview background color and text.

Solution:

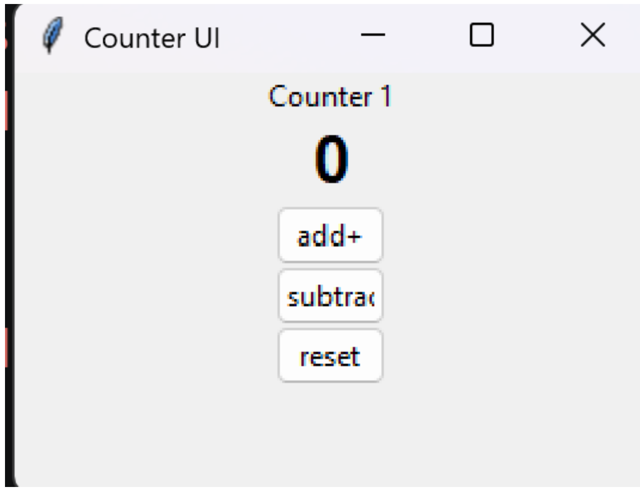
```
import tkinter as tk


def set_preview(bg, text):
    preview.config(bg=bg)
    preview_label.config(text=text, bg=bg)


root = tk.Tk()
root.title("Two Frames")

left = tk.Frame(root)
left.pack(side="left", padx=10, pady=10)

right = tk.Frame(root)
right.pack(side="right", padx=10, pady=10, fill="both", expand=True)

preview = tk.Frame(right, width=300, height=200, bg="white")
preview.pack(fill="both", expand=True)
preview.pack_propagate(False)

preview_label = tk.Label(preview, text="Preview", bg="white", font=("Arial", 16))
preview_label.pack(expand=True)

tk.Button(left, text="Blue", width=12, command=lambda: set_preview("#b3d9ff", "Blue
Theme")).pack(pady=4)
tk.Button(left, text="Green", width=12, command=lambda: set_preview("#b6f2c2", "Green
Theme")).pack(pady=4)
tk.Button(left, text="Yellow", width=12, command=lambda: set_preview("#fff3b0", "Yellow
Theme")).pack(pady=4)
tk.Button(left, text="Reset", width=12, command=lambda: set_preview("white",
"Preview")).pack(pady=8)

root.mainloop()
```



Q16: Create a class Counter that starts at 0. Methods: increment() , decrement() (don’t go below 0), reset() . Create ui with two label and three button .

```python
# counter.py
class Counter:
    def __init__(self):
        self.value = 0

    def increment(self):
        self.value += 1

    def decrement(self):
        if self.value > 0:
            self.value -= 1

    def reset(self):
        self.value = 0
```

```python
# ui.py
import tkinter as tk
from tkinter import ttk
from counter import Counter

root = tk.Tk()
root.title("Counter UI")
root.geometry("200x320")

c1 = Counter()

v1 = tk.StringVar(value=str(c1.value))

def refresh():
      v1.set(str(c1.value))
def inc1():
      c1.increment()
      refresh()

def dec1():
      c1.decrement()
      refresh()

def reset1():
      c1.reset()
      refresh()

ttk.Label(root, text="Counter 1").pack()
ttk.Label(root, textvariable=v1, font=("TkDefaultFont", 20, "bold")).pack()
ttk.Button(root, text="add+", width=6, command=inc1).pack()
ttk.Button(root, text="subtract-", width=6, command=dec1).pack()
ttk.Button(root, text="reset", width=6, command=reset1).pack()

root.mainloop()
```



Q17: Create a class Student with attributes name , id , grades (list). Add methods: add_grade(g) , average() , and is_passed() (pass if average ≥ 50). Write code to test it with 3 students.

```
import tkinter as tk
from tkinter import messagebox

from student import Student

root = tk.Tk()
root.title("Student Information")
root.geometry("520x360")

def getstudentaverage():
      name = entryname.get().strip()
      student_id = entryid.get().strip()
      gr = entrygrades.get().strip()

      if not name or not student_id:
            messagebox.showwarning("Warning", "Please enter student name and ID")
            return
      if not gr:
            messagebox.showwarning("Warning", "Please enter grades separated by comma")
            return

      try:
            grades = [int(g.strip()) for g in gr.split(",")]
      except ValueError:
            messagebox.showerror("Error", "Grades must be numbers separated by commas (example: 80, 
90, 75)")
            return

      s1 = Student(name, student_id, grades)
      lblaverage.config(text=f"Average: {s1.average():.2f}")



lblstudent = tk.Label(root, text="Student Information")
lblstudent.pack(pady=5)
```

```
entryname = tk.Entry(root, width=20)
entryname.pack()

lblid = tk.Label(root, text="Student ID")
lblid.pack(pady=5)

entryid = tk.Entry(root, width=20)
entryid.pack()

lblgrades = tk.Label(root, text="Grades (comma-separated)")
lblgrades.pack(pady=5)

entrygrades = tk.Entry(root, width=20)
entrygrades.pack()
#btn
btnadd = tk.Button(root, text="Calculate", command=getstudentaverage)
btnadd.pack(pady=5)
#label
lblaverage = tk.Label(root, text="Average: -")
lblaverage.pack(pady=5)
root.mainloop()
```