Tkinter GUI Programming

A Comprehensive Guide to Python's Built-in GUI Library

Learn to Build Desktop Applications with Python

What is Tkinter?

- **Tk interface** - Python's standard GUI (Graphical User Interface) toolkit
- **Built-in** - Comes pre-installed with Python
- **Cross-platform** - Works on Windows, macOS, and Linux
- **Lightweight** - Simple and fast for basic to intermediate GUIs
- **Mature** - Been around since 1991, stable and well-documented

```
import tkinter as tk
# That's it! Ready to use.
```

Why Use Tkinter?

Advantages:

- ✅ No installation required
- ✅ Easy to learn for beginners
- ✅ Good for small to medium applications
- ✅ Excellent documentation
- ✅ Large community support

Best For:

- Desktop utilities and tools
- Data entry applications
- Simple games

Your First Tkinter Window

```
import tkinter as tk

# Create the main window
root = tk.Tk()
root.title("My First GUI")
root.geometry("400x300")

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

Key Concepts:

- `tk.Tk()` creates the main window
- `geometry()` sets window size (width x height)
- `mainloop()` starts the event loop (keeps window open)

Basic Widgets Overview

Widget	Purpose	Common Use
`Label`	Display text/images	Show information
`Button`	Clickable button	Trigger actions
`Entry`	Single-line text input	User input fields
`Text`	Multi-line text	Text editors
`Frame`	Container for widgets	Organize layout
`Canvas`	Drawing area	Graphics, games

Label Widget

Display text or images

```
import tkinter as tk

root = tk.Tk()
root.geometry("400x300")

# Simple label
label1 = tk.Label(root, text="Hello, Tkinter!")
label1.pack()

# Styled label
label2 = tk.Label(root,
                  text="Styled Text",
                  font=("Arial", 16, "bold"),
                  fg="blue",
                  bg="yellow")
label2.pack(pady=10)
```

Button Widget

Create clickable buttons

```
import tkinter as tk

def on_click():
    label.config(text="Button was clicked!")

root = tk.Tk()
root.geometry("400x300")

label = tk.Label(root, text="Click the button")
label.pack(pady=20)

button = tk.Button(root,
                   text="Click Me!",
                   command=on_click,
                   bg="green",
                   fg="white",
                   font=("Arial", 12))
button.pack()
```

Entry Widget

Single-line text input

```
import tkinter as tk

def get_input():
    user_text = entry.get()
    label.config(text=f"You entered: {user_text}")

root = tk.Tk()
root.geometry("400x200")

label = tk.Label(root, text="Enter your name:")
label.pack(pady=10)

entry = tk.Entry(root, width=30)
entry.pack(pady=5)

button = tk.Button(root, text="Submit", command=get_input)
button.pack(pady=10)
```

Text Widget

Multi-line text input

```
import tkinter as tk

root = tk.Tk()
root.geometry("500x400")

# Create text widget with scrollbar
text = tk.Text(root, height=15, width=50)
text.pack(side=tk.LEFT, pady=10, padx=10)

scrollbar = tk.Scrollbar(root, command=text.yview)
scrollbar.pack(side=tk.LEFT, fill=tk.Y)

text.config(yscrollcommand=scrollbar.set)

# Insert some text
text.insert(1.0, "Type your text here...\n")
```

Frame Widget

Container for organizing widgets

```
import tkinter as tk

root = tk.Tk()
root.geometry("400x300")

# Top frame
top_frame = tk.Frame(root, bg="lightblue", height=100)
top_frame.pack(fill=tk.X)

tk.Label(top_frame, text="Top Section", bg="lightblue").pack()

# Bottom frame
bottom_frame = tk.Frame(root, bg="lightgreen", height=200)
bottom_frame.pack(fill=tk.BOTH, expand=True)

tk.Label(bottom_frame, text="Bottom Section",
         bg="lightgreen").pack()
```

Layout Managers: Pack

Stacks widgets vertically or horizontally

```
import tkinter as tk

root = tk.Tk()
root.geometry("300x250")

# Pack from top (default)
tk.Label(root, text="Top", bg="red").pack(side=tk.TOP, fill=tk.X)
tk.Label(root, text="Bottom", bg="blue").pack(side=tk.BOTTOM, fill=tk.X)

# Pack from left and right
tk.Label(root, text="Left", bg="green").pack(side=tk.LEFT, fill=tk.Y)
tk.Label(root, text="Right", bg="yellow").pack(side=tk.RIGHT, fill=tk.Y)

# Center (fills remaining space)
tk.Label(root, text="Center", bg="purple").pack(fill=tk.BOTH, expand=True)
root.mainloop()
```

Pack Options

```
widget.pack(
    side=tk.TOP,          # TOP, BOTTOM, LEFT, RIGHT
    fill=tk.NONE,         # NONE, X, Y, BOTH
    expand=False,         # True/False - expand to fill space
    padx=0,              # External horizontal padding
    pady=0,              # External vertical padding
    ipadx=0,             # Internal horizontal padding
    ipady=0              # Internal vertical padding
)
```

Common patterns:

- `pack()` - default, stacks top to bottom
- `pack(side=tk.LEFT)` - arrange left to right
- `pack(fill=tk.BOTH, expand=True)` - fill available space

Layout Managers: Grid

Arrange widgets in rows and columns (like a table)

```
import tkinter as tk

root = tk.Tk()
root.geometry("300x200")

# Create a login form
tk.Label(root, text="Username:").grid(row=0, column=0, sticky=tk.W, padx=10, pady=5)
tk.Entry(root).grid(row=0, column=1, padx=10, pady=5)

tk.Label(root, text="Password:").grid(row=1, column=0, sticky=tk.W, padx=10, pady=5)
tk.Entry(root, show="*").grid(row=1, column=1, padx=10, pady=5)

tk.Button(root, text="Login").grid(row=2, column=0, columnspan=2, pady=10)

root.mainloop()
```

Grid Options

```
widget.grid(
    row=0,              # Row number (0-indexed)
    column=0,           # Column number (0-indexed)
    rowspan=1,          # Span multiple rows
    columnspan=1,       # Span multiple columns
    sticky=tk.W,        # N, S, E, W (compass directions)
    padx=0,            # Horizontal padding
    pady=0             # Vertical padding
)
```

Sticky options:

- `tk.W` - West (left align)
- `tk.E` - East (right align)
- `tk.N+tk.S` - North + South (stretch vertically)
- `tk.E+tk.W` - East + West (stretch horizontally)

Layout Managers: Place

Absolute positioning (use sparingly!)

```
import tkinter as tk

root = tk.Tk()
root.geometry("400x300")

# Absolute positioning (pixels)
tk.Label(root, text="Pixel Position", bg="red").place(x=50, y=50)

# Relative positioning (percentage)
tk.Label(root, text="Center", bg="blue").place(
    relx=0.5,
    rely=0.5,
    anchor=tk.CENTER
)

# Size control
tk.Button(root, text="Fixed Size").place(
    x=100, y=150,
    width=150,
    height=40
)
```

Checkbutton Widget

Create checkboxes

```
import tkinter as tk

root = tk.Tk()
root.geometry("300x250")

# Variables to store checkbox states
var1 = tk.BooleanVar()
var2 = tk.BooleanVar()
var3 = tk.BooleanVar()

tk.Label(root, text="Select your hobbies:").pack(pady=10)

tk.Checkbutton(root, text="Reading", variable=var1).pack(anchor=tk.W, padx=50)
tk.Checkbutton(root, text="Gaming", variable=var2).pack(anchor=tk.W, padx=50)
tk.Checkbutton(root, text="Coding", variable=var3).pack(anchor=tk.W, padx=50)

def show_selection():
    hobbies = []
    if var1.get(): hobbies.append("Reading")
    if var2.get(): hobbies.append("Gaming")
    if var3.get(): hobbies.append("Coding")
    print(f"Selected: {hobbies}")

tk.Button(root, text="Show Selection", command=show_selection).pack(pady=20)
```

Radiobutton Widget

Create radio buttons (single selection)

```
import tkinter as tk

root = tk.Tk()
root.geometry("300x250")

# Variable to store selected option
selection = tk.StringVar(value="Python")

tk.Label(root, text="Choose your favorite language:").pack(pady=10)

tk.Radiobutton(root, text="Python", variable=selection,
               value="Python").pack(anchor=tk.W, padx=50)
tk.Radiobutton(root, text="Java", variable=selection,
               value="Java").pack(anchor=tk.W, padx=50)
tk.Radiobutton(root, text="C++", variable=selection,
               value="C++").pack(anchor=tk.W, padx=50)

def show_choice():
    print(f"Selected: {selection.get()}")
tk.Button(root, text="Submit", command=show_choice).pack(pady=20)
```

Listbox Widget

Scrollable list of items

```
import tkinter as tk

root = tk.Tk()
root.geometry("300x300")

tk.Label(root, text="Select items:").pack(pady=5)

# Create listbox
listbox = tk.Listbox(root, selectmode=tk.MULTIPLE, height=8)
listbox.pack(padx=20, pady=10)

# Add items
items = ["Apple", "Banana", "Cherry", "Date", "Elderberry", "Fig"]
for item in items:
    listbox.insert(tk.END, item)

def get_selected():
    selected = [listbox.get(i) for i in listbox.curselection()]
    print(f"Selected: {selected}")
tk.Button(root, text="Get Selection", command=get_selected).pack(pady=5)
```

Combobox (Dropdown)

From ttk module (themed widgets)

```
import tkinter as tk
from tkinter import ttk

root = tk.Tk()
root.geometry("300x200")

tk.Label(root, text="Select country:").pack(pady=20)

# Create combobox
countries = ["USA", "UK", "Canada", "Australia", "India"]
combo = ttk.Combobox(root, values=countries, state="readonly")
combo.pack(pady=10)
combo.current(0)  # Set default selection

def on_select(event):
    print(f"Selected: {combo.get()}")
combo.bind("<<ComboboxSelected>>", on_select)
```

Scale (Slider) Widget

Create sliders for numeric input

```
import tkinter as tk

root = tk.Tk()
root.geometry("400x300")

value_label = tk.Label(root, text="Value: 50")
value_label.pack(pady=20)

def update_label(val):
    value_label.config(text=f"Value: {int(float(val))}")

# Horizontal slider
scale1 = tk.Scale(root, from_=0, to=100, orient=tk.HORIZONTAL,
                  command=update_label, length=300)
scale1.set(50)
scale1.pack(pady=10)

# Vertical slider
scale2 = tk.Scale(root, from_=0, to=100, orient=tk.VERTICAL, length=150)
scale2.pack(pady=10)
```

Spinbox Widget

Numeric input with up/down buttons

```
import tkinter as tk

root = tk.Tk()
root.geometry("300x200")

tk.Label(root, text="Select quantity:").pack(pady=20)

# Spinbox with range
spinbox1 = tk.Spinbox(root, from_=0, to=100, width=10)
spinbox1.pack(pady=10)

# Spinbox with values
spinbox2 = tk.Spinbox(root, values=("Small", "Medium", "Large"),
                      width=10, state="readonly")
spinbox2.pack(pady=10)

def get_values():
    print(f"Value 1: {spinbox1.get()}")
    print(f"Value 2: {spinbox2.get()}")

tk.Button(root, text="Get Values", command=get_values).pack(pady=20)
```

Message Box Dialogs

Show popup messages

```
import tkinter as tk
from tkinter import messagebox

root = tk.Tk()
root.geometry("300x250")

def show_info():
    messagebox.showinfo("Information", "This is an info message")

def show_warning():
    messagebox.showwarning("Warning", "This is a warning!")

def show_error():
    messagebox.showerror("Error", "An error occurred!")

def ask_question():
    result = messagebox.askyesno("Question", "Do you like Python?")
    print(f"Answer: {'Yes' if result else 'No'}")

tk.Button(root, text="Show Info", command=show_info).pack(pady=5)
tk.Button(root, text="Show Warning", command=show_warning).pack(pady=5)
tk.Button(root, text="Show Error", command=show_error).pack(pady=5)
tk.Button(root, text="Ask Question", command=ask_question).pack(pady=5)
```

File Dialogs

Open and save file dialogs

```
import tkinter as tk
from tkinter import filedialog

root = tk.Tk()
root.geometry("300x200")

def open_file():
    filename = filedialog.askopenfilename(
        title="Select a file",
        filetypes=(("Text files", "*.txt"), ("All files", "*.*"))
    )
    if filename:
        print(f"Selected: {filename}")

def save_file():
    filename = filedialog.asksaveasfilename(
        defaultextension=".txt",
        filetypes=(("Text files", "*.txt"), ("All files", "*.*"))
    )
    if filename:
        print(f"Save as: {filename}")
tk.Button(root, text="Open File", command=open_file).pack(pady=20)
```

Canvas Widget

Draw shapes and create graphics

```
import tkinter as tk

root = tk.Tk()
root.geometry("500x400")

canvas = tk.Canvas(root, width=450, height=350, bg="white")
canvas.pack(pady=10)

# Draw shapes
canvas.create_line(50, 50, 400, 50, fill="blue", width=3)
canvas.create_rectangle(50, 100, 150, 200, fill="red", outline="black")
canvas.create_oval(200, 100, 300, 200, fill="green")
canvas.create_polygon(350, 100, 400, 150, 350, 200, fill="yellow")

# Draw text
canvas.create_text(225, 250, text="Canvas Graphics",
                   font=("Arial", 20, "bold"))
```

Event Binding

Respond to mouse and keyboard events

```
import tkinter as tk

root = tk.Tk()
root.geometry("400x300")

label = tk.Label(root, text="Click or press keys", font=("Arial", 14))
label.pack(pady=50)

def on_click(event):
    label.config(text=f"Clicked at ({event.x}, {event.y})")

def on_key(event):
    label.config(text=f"Key pressed: {event.char}")

def on_enter(event):
    label.config(bg="lightblue")

def on_leave(event):
    label.config(bg="white")

label.bind("<Button-1>", on_click)      # Left click
root.bind("<Key>", on_key)              # Any key
label.bind("<Enter>", on_enter)         # Mouse enter
label.bind("<Leave>", on_leave)         # Mouse leave
```

Common Event Types

Event	Description
`<Button-1>`	Left mouse click
`<Button-3>`	Right mouse click
`<Double-Button-1>`	Double click
`<Motion>`	Mouse movement
`<Key>`	Any key press
`<Return>`	Enter key
`<space>`	Space bar
`<Enter>`	Mouse enters widget
`<Leave>`	Mouse leaves widget

Menu Bar

Create application menus

```
import tkinter as tk

root = tk.Tk()
root.geometry("400x300")

# Create menu bar
menubar = tk.Menu(root)
root.config(menu=menubar)

# File menu
file_menu = tk.Menu(menubar, tearoff=0)
menubar.add_cascade(label="File", menu=file_menu)
file_menu.add_command(label="New", command=lambda: print("New"))
file_menu.add_command(label="Open", command=lambda: print("Open"))
file_menu.add_separator()
file_menu.add_command(label="Exit", command=root.quit)

# Edit menu
edit_menu = tk.Menu(menubar, tearoff=0)
menubar.add_cascade(label="Edit", menu=edit_menu)
edit_menu.add_command(label="Cut", command=lambda: print("Cut"))
edit_menu.add_command(label="Copy", command=lambda: print("Copy"))
edit_menu.add_command(label="Paste", command=lambda: print("Paste"))
```

Tkinter Variables

Special variables that auto-update widgets

```
import tkinter as tk

root = tk.Tk()
root.geometry("300x200")

# Create tkinter variables
string_var = tk.StringVar(value="Hello")
int_var = tk.IntVar(value=42)
bool_var = tk.BooleanVar(value=True)
double_var = tk.DoubleVar(value=3.14)

# Link variables to widgets
entry = tk.Entry(root, textvariable=string_var)
entry.pack(pady=10)

label = tk.Label(root, textvariable=string_var)
label.pack(pady=10)

def update():
    string_var.set("Updated!")

tk.Button(root, text="Update", command=update).pack(pady=10)
```

Widget Configuration

Modify widget properties dynamically

```
import tkinter as tk

root = tk.Tk()
root.geometry("300x250")

label = tk.Label(root, text="Original Text", font=("Arial", 14))
label.pack(pady=20)

button = tk.Button(root, text="Click Me")
button.pack(pady=10)

def change_label():
    label.config(text="Text Changed!", fg="red", bg="yellow")

def change_button():
    button.config(text="Clicked!", bg="green", fg="white", state=tk.DISABLED)

def reset():
    label.config(text="Original Text", fg="black", bg="white")
    button.config(text="Click Me", bg="SystemButtonFace", state=tk.NORMAL)

button.config(command=change_button)
tk.Button(root, text="Change Label", command=change_label).pack(pady=5)
```

Widget States

Control widget behavior

```
import tkinter as tk

root = tk.Tk()
root.geometry("300x200")

entry = tk.Entry(root)
entry.pack(pady=20)

button = tk.Button(root, text="Submit")
button.pack(pady=10)

def toggle_widgets():
    # Get current state
    current_state = str(entry.cget('state'))

    if current_state == 'normal':
        entry.config(state=tk.DISABLED)
        button.config(state=tk.DISABLED)
        toggle_btn.config(text="Enable Widgets")
    else:
        entry.config(state=tk.NORMAL)
        button.config(state=tk.NORMAL)
        toggle_btn.config(text="Disable Widgets")
toggle_btn = tk.Button(root, text="Disable Widgets", command=toggle_widgets)
```

Styling with ttk

Themed widgets with better appearance

```
import tkinter as tk
from tkinter import ttk

root = tk.Tk()
root.geometry("400x300")

# Standard tkinter widgets (left side)
frame1 = tk.Frame(root, bg="lightgray")
frame1.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=5)

tk.Label(frame1, text="Standard Tkinter", bg="lightgray").pack(pady=10)
tk.Button(frame1, text="Button").pack(pady=5)
tk.Entry(frame1).pack(pady=5)

# ttk themed widgets (right side)
frame2 = tk.Frame(root)
frame2.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True, padx=5)

ttk.Label(frame2, text="Themed ttk").pack(pady=10)
ttk.Button(frame2, text="Button").pack(pady=5)
ttk.Entry(frame2).pack(pady=5)
```

Progressbar Widget

Show progress of operations

```
import tkinter as tk
from tkinter import ttk
import time

root = tk.Tk()
root.geometry("400x200")

# Determinate progress bar
progress1 = ttk.Progressbar(root, length=300, mode='determinate')
progress1.pack(pady=20)

# Indeterminate progress bar
progress2 = ttk.Progressbar(root, length=300, mode='indeterminate')
progress2.pack(pady=20)

def start_progress():
    progress2.start(10)  # Update every 10ms
    for i in range(101):
        progress1['value'] = i
        root.update()
        time.sleep(0.02)
    progress2.stop()

tk.Button(root, text="Start Progress", command=start_progress).pack(pady=20)
```

Notebook (Tabs)

Create tabbed interfaces

```
import tkinter as tk
from tkinter import ttk

root = tk.Tk()
root.geometry("400x300")

# Create notebook
notebook = ttk.Notebook(root)
notebook.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

# Create tabs
tab1 = tk.Frame(notebook)
tab2 = tk.Frame(notebook)
tab3 = tk.Frame(notebook)

notebook.add(tab1, text="Home")
notebook.add(tab2, text="Settings")
notebook.add(tab3, text="About")

# Add content to tabs
tk.Label(tab1, text="Welcome to the Home tab!", font=("Arial", 14)).pack(pady=50)
tk.Label(tab2, text="Settings go here", font=("Arial", 14)).pack(pady=50)
tk.Label(tab3, text="About this app", font=("Arial", 14)).pack(pady=50)
```

Treeview Widget

Display hierarchical data

```
import tkinter as tk
from tkinter import ttk

root = tk.Tk()
root.geometry("500x300")

# Create treeview
tree = ttk.Treeview(root, columns=("Age", "Country"), show="tree headings")
tree.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

# Define columns
tree.heading("#0", text="Name")
tree.heading("Age", text="Age")
tree.heading("Country", text="Country")

# Add data
tree.insert("", tk.END, text="Alice", values=(25, "USA"))
tree.insert("", tk.END, text="Bob", values=(30, "UK"))

# Add child items
parent = tree.insert("", tk.END, text="Team A")
tree.insert(parent, tk.END, text="Charlie", values=(28, "Canada"))
tree.insert(parent, tk.END, text="Diana", values=(32, "Australia"))
```

Window Properties

Customize window behavior

```
import tkinter as tk

root = tk.Tk()

# Window title
root.title("Custom Window")

# Window size and position
root.geometry("400x300+100+100")  # widthxheight+x+y

# Minimum and maximum size
root.minsize(300, 200)
root.maxsize(800, 600)

# Resizable
root.resizable(True, True)  # (width, height)

# Window icon (must be .ico on Windows)
# root.iconbitmap("icon.ico")

# Always on top
root.attributes("-topmost", True)

# Fullscreen
# root.attributes("-fullscreen", True)

# Transparent (0.0 to 1.0)
```

Color Chooser

Let users pick colors

```
import tkinter as tk
from tkinter import colorchooser

root = tk.Tk()
root.geometry("300x200")

label = tk.Label(root, text="Click to choose color",
                 font=("Arial", 14), bg="white")
label.pack(fill=tk.BOTH, expand=True)

def choose_color():
    color = colorchooser.askcolor(title="Choose color")
    if color[1]:  # color[1] is hex code
        label.config(bg=color[1])
        root.config(bg=color[1])
        print(f"RGB: {color[0]}")
        print(f"Hex: {color[1]}")

button = tk.Button(root, text="Choose Color", command=choose_color)
button.pack(pady=20)
```

Font Customization

Working with fonts

```
import tkinter as tk
from tkinter import font

root = tk.Tk()
root.geometry("400x400")

# Get all available fonts
available_fonts = font.families()
print(f"Available fonts: {len(available_fonts)}")

# Different font styles
tk.Label(root, text="Default Font").pack(pady=5)
tk.Label(root, text="Arial 14", font=("Arial", 14)).pack(pady=5)
tk.Label(root, text="Bold", font=("Arial", 14, "bold")).pack(pady=5)
tk.Label(root, text="Italic", font=("Arial", 14, "italic")).pack(pady=5)
tk.Label(root, text="Bold Italic", font=("Arial", 14, "bold italic")).pack(pady=5)
tk.Label(root, text="Underline", font=("Arial", 14, "underline")).pack(pady=5)
tk.Label(root, text="Overstrike", font=("Arial", 14, "overstrike")).pack(pady=5)

# Custom font object
custom_font = font.Font(family="Courier", size=16, weight="bold")
tk.Label(root, text="Custom Font Object", font=custom_font).pack(pady=5)
```

Grid Weight System

Make responsive layouts

```
import tkinter as tk

root = tk.Tk()
root.geometry("400x300")

# Configure grid weights (makes cells expandable)
root.grid_rowconfigure(0, weight=1)
root.grid_rowconfigure(1, weight=2)  # Row 1 is 2x taller
root.grid_columnconfigure(0, weight=1)
root.grid_columnconfigure(1, weight=2)  # Column 1 is 2x wider

# Add widgets
tk.Label(root, text="Top Left", bg="red").grid(
    row=0, column=0, sticky="nsew", padx=2, pady=2)
tk.Label(root, text="Top Right", bg="blue").grid(
    row=0, column=1, sticky="nsew", padx=2, pady=2)
tk.Label(root, text="Bottom Left", bg="green").grid(
    row=1, column=0, sticky="nsew", padx=2, pady=2)
tk.Label(root, text="Bottom Right", bg="yellow").grid(
    row=1, column=1, sticky="nsew", padx=2, pady=2)
```

LabelFrame Widget

Grouped widgets with borders

```
import tkinter as tk

root = tk.Tk()
root.geometry("400x300")

# Personal Info group
personal_frame = tk.LabelFrame(root, text="Personal Information",
                               font=("Arial", 10, "bold"))
personal_frame.pack(padx=20, pady=10, fill="both")

tk.Label(personal_frame, text="Name:").grid(row=0, column=0, sticky="w", padx=5, pady=5)
tk.Entry(personal_frame).grid(row=0, column=1, padx=5, pady=5)

tk.Label(personal_frame, text="Age:").grid(row=1, column=0, sticky="w", padx=5, pady=5)
tk.Entry(personal_frame).grid(row=1, column=1, padx=5, pady=5)

# Contact Info group
contact_frame = tk.LabelFrame(root, text="Contact Information",
                              font=("Arial", 10, "bold"))
contact_frame.pack(padx=20, pady=10, fill="both")

tk.Label(contact_frame, text="Email:").grid(row=0, column=0, sticky="w", padx=5, pady=5)
tk.Entry(contact_frame).grid(row=0, column=1, padx=5, pady=5)
```

Best Practice: Separation of Concerns

Organize code with classes

```
import tkinter as tk

class Application(tk.Tk):
    def __init__(self):
        super().__init__()

        self.title("My Application")
        self.geometry("400x300")

        self.create_widgets()

    def create_widgets(self):
        self.label = tk.Label(self, text="Hello, World!",
                              font=("Arial", 16))
        self.label.pack(pady=20)

        self.button = tk.Button(self, text="Click Me",
                                command=self.on_button_click)
        self.button.pack(pady=10)

    def on_button_click(self):
        self.label.config(text="Button Clicked!")

if __name__ == "__main__":
```

Complete Example: TODO App

```
import tkinter as tk
from tkinter import messagebox

class TodoApp:
    def __init__(self, root):
        self.root = root
        self.root.title("TODO List")
        self.root.geometry("400x500")

        # Entry
        self.entry = tk.Entry(root, font=("Arial", 14))
        self.entry.pack(pady=10, padx=10, fill=tk.X)
        self.entry.bind("<Return>", lambda e: self.add_task())

        # Buttons frame
        btn_frame = tk.Frame(root)
        btn_frame.pack(pady=5)

        tk.Button(btn_frame, text="Add Task", command=self.add_task,
                  bg="green", fg="white").pack(side=tk.LEFT, padx=5)
        tk.Button(btn_frame, text="Delete Task", command=self.delete_task,
                  bg="red", fg="white").pack(side=tk.LEFT, padx=5)

        # Listbox
        self.listbox = tk.Listbox(root, font=("Arial", 12))
        self.listbox.pack(pady=10, padx=10, fill=tk.BOTH, expand=True)

    def add_task(self):
        task = self.entry.get()
        if task:
            self.listbox.insert(tk.END, task)
            self.entry.delete(0, tk.END)
        else:
            messagebox.showwarning("Warning", "Please enter a task!")

    def delete_task(self):
        try:
            self.listbox.delete(self.listbox.curselection())
        except:
            messagebox.showwarning("Warning", "Please select a task!")

root = tk.Tk()
app = TodoApp(root)
root.mainloop()
```

Complete Example: Calculator

```
import tkinter as tk

class Calculator:
    def __init__(self, root):
        self.root = root
        self.root.title("Calculator")
        self.root.geometry("300x400")
        self.root.resizable(False, False)

        self.expression = ""

        # Display
        self.display = tk.Entry(root, font=("Arial", 20),
                                justify="right", bd=10)
        self.display.grid(row=0, column=0, columnspan=4,
                          sticky="nsew", padx=5, pady=5)

        # Buttons
        buttons = [
            '7', '8', '9', '/',
            '4', '5', '6', '*',
            '1', '2', '3', '-',
            'C', '0', '=', '+'
        ]

        row = 1
        col = 0
        for button in buttons:
            cmd = lambda x=button: self.click(x)
            tk.Button(root, text=button, font=("Arial", 18),
                      command=cmd).grid(row=row, column=col,
                                        sticky="nsew", padx=2, pady=2)
            col += 1
            if col > 3:
                col = 0
                row += 1

        # Configure grid
        for i in range(5):
            root.grid_rowconfigure(i, weight=1)
        for i in range(4):
            root.grid_columnconfigure(i, weight=1)

    def click(self, key):
        if key == '=':
            try:
                result = eval(self.expression)
                self.display.delete(0, tk.END)
                self.display.insert(0, str(result))
                self.expression = str(result)
            except:
                self.display.delete(0, tk.END)
                self.display.insert(0, "Error")
                self.expression = ""
        elif key == 'C':
            self.expression = ""
            self.display.delete(0, tk.END)
        else:
            self.expression += str(key)
            self.display.delete(0, tk.END)
            self.display.insert(0, self.expression)

root = tk.Tk()
app = Calculator(root)
root.mainloop()
```

Threading with Tkinter

Keep GUI responsive during long operations

```
import tkinter as tk
import threading
import time

def long_operation():
    # Simulate long task
    for i in range(10):
        time.sleep(0.5)
        # Update GUI from thread (not recommended directly)
        # Use queue or after() instead
        root.after(0, lambda: progress_label.config(
            text=f"Progress: {(i+1)*10}%"))
    root.after(0, lambda: progress_label.config(text="Complete!"))
    root.after(0, lambda: button.config(state=tk.NORMAL))

root = tk.Tk()
root.geometry("300x150")

progress_label = tk.Label(root, text="Ready", font=("Arial", 14))
progress_label.pack(pady=30)

def start_task():
    button.config(state=tk.DISABLED)
    progress_label.config(text="Working...")
    thread = threading.Thread(target=long_operation, daemon=True)
    thread.start()

button = tk.Button(root, text="Start Long Task", command=start_task)
```

Custom Widgets

Create reusable components

```
import tkinter as tk

class LabeledEntry(tk.Frame):
    def __init__(self, parent, label_text, **kwargs):
        super().__init__(parent)

        self.label = tk.Label(self, text=label_text, width=15, anchor='w')
        self.label.pack(side=tk.LEFT, padx=(0, 10))

        self.entry = tk.Entry(self, **kwargs)
        self.entry.pack(side=tk.LEFT, fill=tk.X, expand=True)

    def get(self):
        return self.entry.get()

    def set(self, value):
        self.entry.delete(0, tk.END)
        self.entry.insert(0, value)

# Usage
root = tk.Tk()
root.geometry("400x200")

name_field = LabeledEntry(root, "Name:")
name_field.pack(padx=20, pady=10, fill=tk.X)

email_field = LabeledEntry(root, "Email:")
email_field.pack(padx=20, pady=10, fill=tk.X)

def get_data():
    print(f"Name: {name_field.get()}")
    print(f"Email: {email_field.get()}")
```

Error Handling

Handle exceptions gracefully

```
import tkinter as tk
from tkinter import messagebox

def safe_divide():
    try:
        num1 = float(entry1.get())
        num2 = float(entry2.get())
        result = num1 / num2
        result_label.config(text=f"Result: {result:.2f}", fg="green")
    except ValueError:
        messagebox.showerror("Error", "Please enter valid numbers!")
    except ZeroDivisionError:
        messagebox.showerror("Error", "Cannot divide by zero!")
    except Exception as e:
        messagebox.showerror("Error", f"An error occurred: {str(e)}")

root = tk.Tk()
root.geometry("300x250")

tk.Label(root, text="Number 1:").pack(pady=5)
entry1 = tk.Entry(root)
entry1.pack(pady=5)

tk.Label(root, text="Number 2:").pack(pady=5)
entry2 = tk.Entry(root)
entry2.pack(pady=5)

tk.Button(root, text="Divide", command=safe_divide).pack(pady=10)

result_label = tk.Label(root, text="Result: ", font=("Arial", 12))
```

Validation

Validate user input

```
import tkinter as tk

root = tk.Tk()
root.geometry("300x200")

def validate_number(char):
    return char.isdigit() or char == ""

def validate_age(value):
    if value == "":
        return True
    try:
        age = int(value)
        return 0 <= age <= 120
    except:
        return False

# Register validation functions
vcmd_num = (root.register(validate_number), '%S')
vcmd_age = (root.register(validate_age), '%P')

tk.Label(root, text="Phone (digits only):").pack(pady=10)
phone_entry = tk.Entry(root, validate="key", validatecommand=vcmd_num)
phone_entry.pack(pady=5)

tk.Label(root, text="Age (0-120):").pack(pady=10)
age_entry = tk.Entry(root, validate="key", validatecommand=vcmd_age)
```

Tooltips

Add hover tooltips to widgets

```
import tkinter as tk

class ToolTip:
    def __init__(self, widget, text):
        self.widget = widget
        self.text = text
        self.tooltip = None
        widget.bind("<Enter>", self.show)
        widget.bind("<Leave>", self.hide)

    def show(self, event):
        x, y, _, _ = self.widget.bbox("insert")
        x += self.widget.winfo_rootx() + 25
        y += self.widget.winfo_rooty() + 25

        self.tooltip = tk.Toplevel(self.widget)
        self.tooltip.wm_overrideredirect(True)
        self.tooltip.wm_geometry(f"+{x}+{y}")

        label = tk.Label(self.tooltip, text=self.text,
                         background="yellow", relief=tk.SOLID, borderwidth=1)
        label.pack()

    def hide(self, event):
        if self.tooltip:
            self.tooltip.destroy()
            self.tooltip = None

root = tk.Tk()
root.geometry("300x200")

button1 = tk.Button(root, text="Hover over me")
button1.pack(pady=20)
ToolTip(button1, "This is a helpful tooltip!")

button2 = tk.Button(root, text="Me too!")
```

Drag and Drop

Implement draggable widgets

```
import tkinter as tk

class DraggableLabel:
    def __init__(self, parent, text):
        self.label = tk.Label(parent, text=text, bg="lightblue",
                              relief=tk.RAISED, padx=20, pady=10)
        self.label.place(x=50, y=50)

        self.label.bind("<Button-1>", self.start_drag)
        self.label.bind("<B1-Motion>", self.drag)

        self.start_x = 0
        self.start_y = 0

    def start_drag(self, event):
        self.start_x = event.x
        self.start_y = event.y

    def drag(self, event):
        x = self.label.winfo_x() + event.x - self.start_x
        y = self.label.winfo_y() + event.y - self.start_y
        self.label.place(x=x, y=y)

root = tk.Tk()
root.geometry("400x300")
DraggableLabel(root, "Drag me!")
```

Keyboard Shortcuts

Add hotkeys to your application

```
import tkinter as tk

root = tk.Tk()
root.geometry("400x300")

status = tk.Label(root, text="Press a hotkey", font=("Arial", 14))
status.pack(pady=50)

def new_file(event=None):
    status.config(text="New File (Ctrl+N)")

def open_file(event=None):
    status.config(text="Open File (Ctrl+O)")

def save_file(event=None):
    status.config(text="Save File (Ctrl+S)")

def quit_app(event=None):
    root.quit()

# Bind keyboard shortcuts
root.bind("<Control-n>", new_file)
root.bind("<Control-o>", open_file)
root.bind("<Control-s>", save_file)
root.bind("<Control-q>", quit_app)

# Display shortcuts
info = """
Keyboard Shortcuts:
Ctrl+N: New File
Ctrl+O: Open File
Ctrl+S: Save File
Ctrl+Q: Quit
"""
```

Animated Widgets

Create simple animations

```
import tkinter as tk

root = tk.Tk()
root.geometry("400x400")

canvas = tk.Canvas(root, width=400, height=400, bg="white")
canvas.pack()

# Create animated circle
circle = canvas.create_oval(175, 175, 225, 225, fill="red")

x_speed = 5
y_speed = 3

def animate():
    global x_speed, y_speed

    # Get current position
    coords = canvas.coords(circle)

    # Move circle
    canvas.move(circle, x_speed, y_speed)

    # Bounce off walls
    if coords[0] <= 0 or coords[2] >= 400:
        x_speed = -x_speed
    if coords[1] <= 0 or coords[3] >= 400:
        y_speed = -y_speed
    # Schedule next frame
    root.after(20, animate)
```

Common Pitfalls to Avoid

1. **Mixing layout managers** in the same container

```
# DON'T DO THIS
frame.pack()
frame.grid()  # Error!
```

2. **Forgetting mainloop()**

```
# Window won't show!
root = tk.Tk()
# root.mainloop()  # Missing!
```

3. **Using pack/grid return value**

```
# DON'T DO THIS
button = tk.Button(root, text="Click").pack()  # button is None!
```

Performance Tips

1. **Use update_idletasks() instead of update()**

```
# Better performance
root.update_idletasks()  # Only updates geometry
```

2. **Batch canvas operations**

```
# Faster
canvas.itemconfig("all", state=tk.HIDDEN)
# Than individual updates
```

3. **Use after() for scheduled tasks**

```
# Instead of time.sleep() in loop
def update():
    # Do work
    root.after(100, update)  # Schedule next update
```

Debugging Tkinter

Useful techniques for troubleshooting

```
import tkinter as tk

root = tk.Tk()

# Print all widget info
def debug_widget(widget, level=0):
    print("  " * level + str(widget))
    for child in widget.winfo_children():
        debug_widget(child, level + 1)

# Get widget under mouse
def widget_under_mouse(event):
    x, y = root.winfo_pointerx(), root.winfo_pointery()
    widget = root.winfo_containing(x, y)
    print(f"Widget: {widget}")

# Print all keycodes
def print_event(event):
    print(f"Key: {event.keysym}, Code: {event.keycode}, Char: {event.char}")

root.bind("<Key>", print_event)
root.bind("<Button-3>", widget_under_mouse)
tk.Button(root, text="Debug",
```

Resources for Learning More

Official Documentation:

- Python Tkinter docs: docs.python.org/3/library/tkinter.html
- Tcl/Tk documentation: tcl.tk/man/

Tutorials & Guides:

- Real Python Tkinter tutorials
- TutorialsPoint Tkinter guide
- GeeksforGeeks Tkinter articles

Books:

- "Python GUI Programming with Tkinter" by Alan D. Moore
- "Programming Python" by Mark Lutz

Summary: Key Takeaways

✅ **Tkinter is perfect for:**

- Learning GUI programming
- Quick prototypes and tools
- Small to medium desktop apps
- Educational projects

✅ **Remember:**

- Always call `mainloop()`
- Don't mix layout managers in same parent
- Use classes for complex applications
- Handle errors gracefully

Thank You!

Questions?

Start building your GUI applications today!

```
import tkinter as tk

root = tk.Tk()
root.title("Your Awesome App")
root.geometry("600x400")

tk.Label(root, text="Happy Coding!",
         font=("Arial", 24, "bold")).pack(expand=True)

root.mainloop()
```

Good luck with your Tkinter projects! 🚀