GUIs, Data Display & Event-Driven Programming

Computer Science

November 29, 2025

What is a GUI?

- **GUI** = Graphical User Interface
- Visual way to interact with computers
- Uses windows, icons, buttons, menus
- Instead of typing commands

Real Example

- Web browsers (Chrome, Firefox)
- Mobile apps
- Video games
- ATM machines

Why Use GUIs?

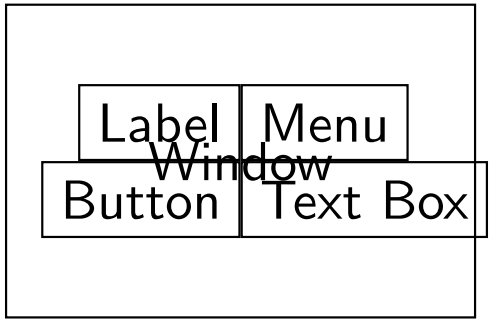
Command Line

- Text-based
- Requires memorization
- Less intuitive
- Faster for experts

GUI

- Visual interface
- Easy to learn
- User-friendly
- Accessible to everyone

Common GUI Components



- **Buttons**: Click to perform actions
- **Text Fields**: Input text
- **Labels**: Display text
- **Menus**: List of options
- **Checkboxes**: Multiple selections

Popular GUI Frameworks

Python

- Tkinter
- PyQt
- Kivy
- wxPython

Java

- Swing
- JavaFX
- AWT

Web

- HTML/CSS/JS
- React
- Angular
- Vue.js

Tkinter - Python GUI Framework

Why Tkinter?

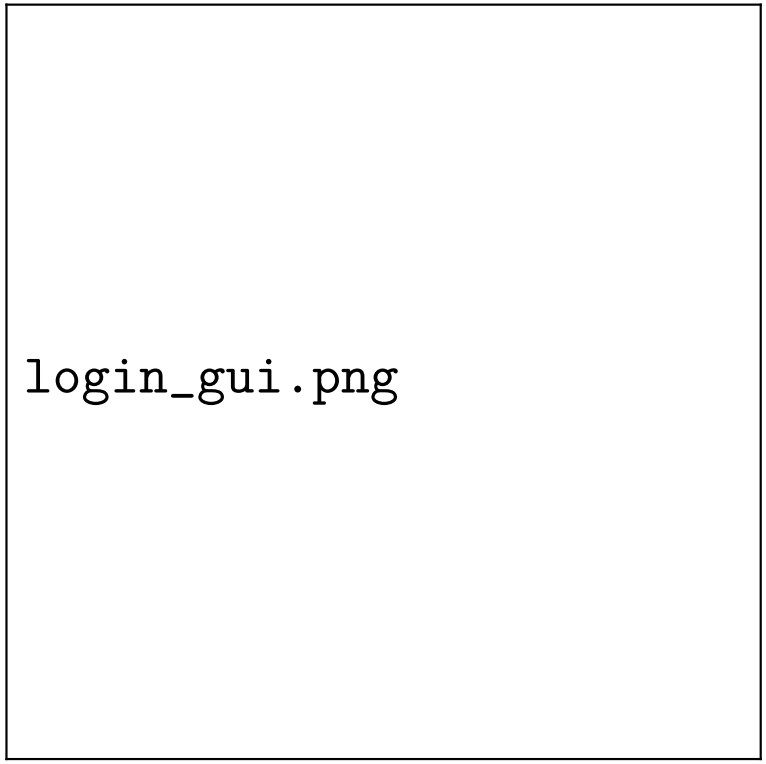
- Built into Python
- Easy to learn
- Cross-platform
- Good for beginners

Simple Tkinter App

Your First GUI: Login Screen

Components Needed:

- Window
- Username label
- Username entry
- Password label
- Password entry
- Login button
- Status label



Login Screen Code

```
import tkinter as tk
from tkinter import messagebox

def login():
    username = entry_username.get()
    password = entry_password.get()

    if username == "admin" and password == "password":
        messagebox.showinfo("Login", "Success!")
    else:
        messagebox.showerror("Login", "Failed!")

# Create main window
root = tk.Tk()
root.title("Login System")
root.geometry("300x200")

# Create widgets
label_username = tk.Label(root, text="Username:")
label_password = tk.Label(root, text="Password:")
entry_username = tk.Entry(root)
```

GUI Layout Managers

pack()

- Simple packing
- Automatic placement
- Good for simple layouts

grid()

- Table-like layout
- Rows and columns
- Most flexible

place()

- Absolute positioning
- Pixel coordinates
- Rarely used

Ways to Display Data in GUIs

- **Labels**: Simple text display
- **Text Widgets**: Multi-line text
- **Tables/Grids**: Structured data
- **Charts/Graphs**: Visual data representation
- **Progress Bars**: Show progress
- **Listboxes**: List of items

Student Grade Display

```
import tkinter as tk
from tkinter import ttk

def show_grades():
    # Sample student data
    students = [
        {"Name": "Alice", "Math": 85, "Science": 92},
        {"Name": "Bob", "Math": 78, "Science": 88},
        {"Name": "Charlie", "Math": 92, "Science": 85}
    ]

    # Clear previous data
    for row in tree.get_children():
        tree.delete(row)

    # Insert new data
    for student in students:
        tree.insert("", "end", values=(
            student["Name"],
            student["Math"],
            student["Science"]
```

Data Visualization in GUIs

charts.png

Simple Chart in GUI

```
import tkinter as tk
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import
    FigureCanvasTkAgg

def create_chart():
    # Sample data
    subjects = ['Math', 'Science', 'English', 'History']
    grades = [85, 92, 78, 88]

    # Create figure
    fig, ax = plt.subplots()
    ax.bar(subjects, grades, color=['red', 'blue', 'green',
        'orange'])
    ax.set_ylabel('Grades')
    ax.set_title('Student Performance')

    # Embed in Tkinter
    canvas = FigureCanvasTkAgg(fig, master=root)
    canvas.draw()
    canvas.get_tk_widget().pack()
```

What is Event-Driven Programming?

Definition

Programming paradigm where the flow of the program is determined by events

- **Events**: User actions or system occurrences
- **Event Handlers**: Functions that respond to events
- **Event Loop**: Constantly checks for new events

Real World Analogy

Like a restaurant:

- Customer orders (event)
- Kitchen prepares food (event handler)
- Waiter serves food (response)

Common GUI Events

Mouse Events

- Click
- Double-click
- Mouse move
- Drag and drop
- Scroll

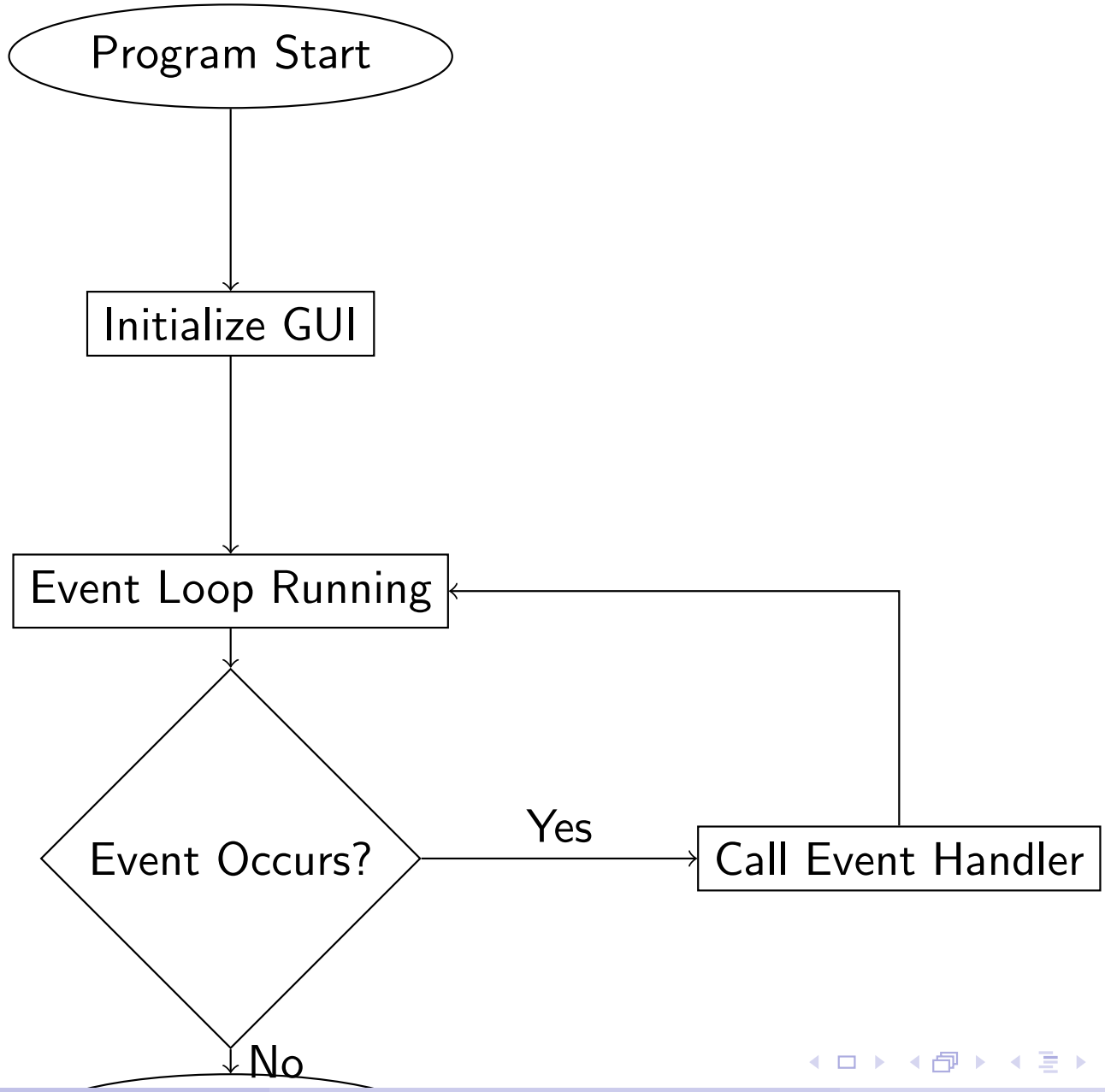
Keyboard Events

- Key press
- Key release
- Special keys (Enter, Esc)

Window Events

- Resize
- Close
- Minimize

Event-Driven Programming Flow



Event Handling Example

```
import tkinter as tk

class EventDemo:
    def __init__(self):
        self.root = tk.Tk()
        self.root.title("Event Demo")

        # Create widgets
        self.button = tk.Button(self.root, text="Click Me!")
        self.label = tk.Label(self.root, text="No events yet
            ")
        self.entry = tk.Entry(self.root)

        # Bind events
        self.button.bind("<Button-1>", self.button_click)
        self.entry.bind("<KeyRelease>", self.key_pressed)
        self.root.bind("<Configure>", self.window_resized)

        # Layout
        self.button.pack()
        self.entry.pack()
```

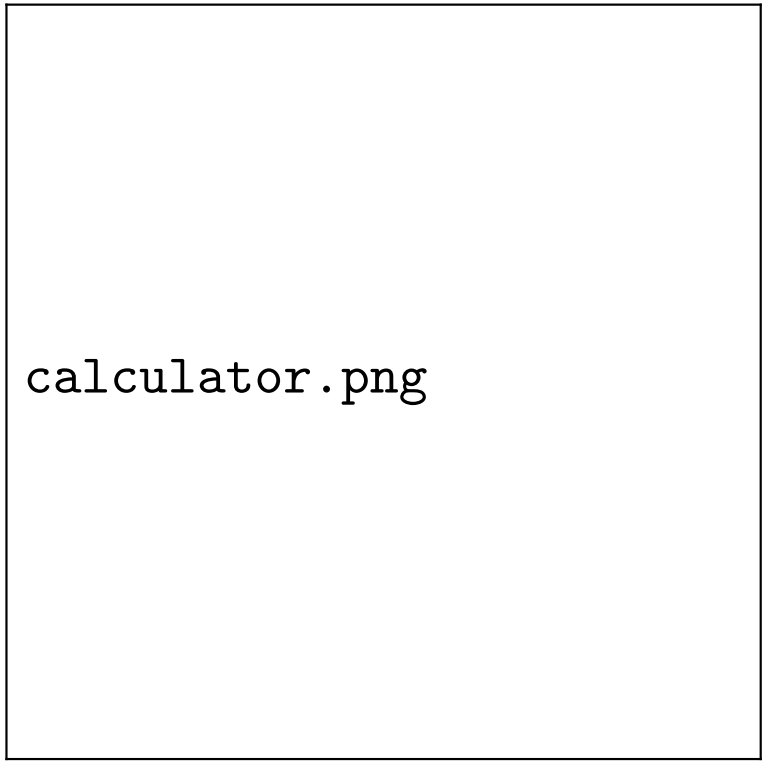
Example 1: Calculator App

Features:

- Number buttons (0-9)
- Operation buttons (+, -, *, /)
- Display screen
- Clear button
- Equals button

Events:

- Button clicks
- Keyboard input



Calculator Implementation

```
import tkinter as tk

class Calculator:
    def __init__(self):
        self.root = tk.Tk()
        self.root.title("Calculator")
        self.display = tk.Entry(self.root, width=20, font=('
            Arial', 16))
        self.display.grid(row=0, column=0, columnspan=4)
        self.create_buttons()
        self.current_input = ""

    def create_buttons(self):
        buttons = [
            '7', '8', '9', '/',
            '4', '5', '6', '*',
            '1', '2', '3', '-',
            '0', 'C', '=', '+'
        ]

        row, col = 1, 0
```

Example 2: Weather App

Features:

- City input field
- Search button
- Temperature display
- Weather description
- Humidity/Wind speed
- Weather icon

Events:

- Button click
- Enter key press
- Data loading completion

Example 3: To-Do List App

Features:

- Add new tasks
- Mark tasks complete
- Delete tasks
- Filter tasks
- Save/load tasks

Events:

- Add button click
- Checkbox toggle
- Delete button click
- Enter key in input
- Window close (save)

To-Do List Implementation

```
import tkinter as tk
from tkinter import messagebox

class TodoApp:
    def __init__(self):
        self.root = tk.Tk()
        self.root.title("To-Do List")
        self.tasks = []

        # Create widgets
        self.task_entry = tk.Entry(self.root, width=30)
        self.add_button = tk.Button(self.root, text="Add
            Task",
                                    command=self.add_task)
        self.task_list = tk.Listbox(self.root, width=50,
            height=15)
        self.delete_button = tk.Button(self.root, text="
            Delete Selected",
                                       command=self.
                                           delete_task)
```

MVC Architecture in GUIs

Model

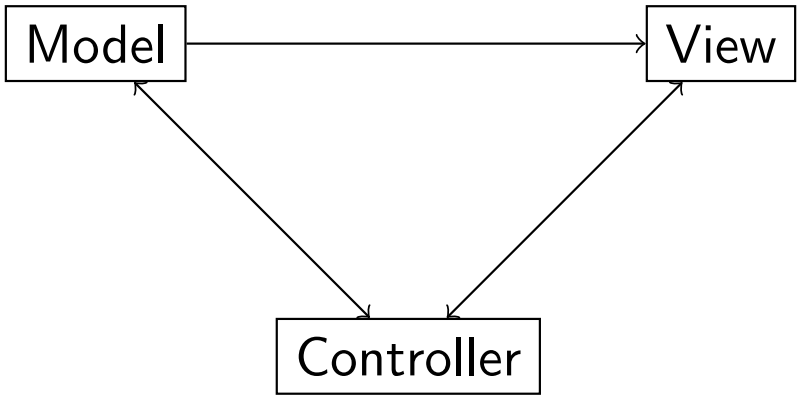
- Data and logic
- Doesn't know about GUI
- Manages application state

View

- GUI presentation
- Displays data to user
- Sends user actions to controller

Controller

- Handles user input
- Updates model
- Refreshes view



Threading in GUIs

Important Rule

Never block the main GUI thread with long operations!

- **Problem**: Long operations freeze the GUI
- **Solution**: Use threads for background tasks
- **Examples**:
  - Downloading files
  - Processing large data
  - Database operations
  - Web requests

Threading Example

```
import tkinter as tk
import threading
import time

class ThreadedApp:
    def __init__(self):
        self.root = tk.Tk()
        self.root.title("Threading Example")

        self.progress = tk.Label(self.root, text="Ready")
        self.start_button = tk.Button(self.root, text="Start
            Long Task",
                                      command=self.start_task
                                          )
        self.stop_button = tk.Button(self.root, text="Stop",
            state=tk.DISABLED)

        self.progress.pack()
        self.start_button.pack()
        self.stop_button.pack()
```

GUI Design Principles

1. Keep it Simple

- Minimalistic design
- Intuitive layout
- Clear purpose

2. Be Consistent

- Standard controls
- Predictable behavior
- Familiar patterns

3. Provide Feedback

- Loading indicators
- Success/error messages
- Progress updates

Event Handling Best Practices

- **Keep handlers short**: Delegate work to other functions
- **Use descriptive names**: `on_login_click()` not `click1()`
- **Handle errors gracefully**: Don't let crashes affect GUI
- **Update GUI properly**: Use thread-safe methods
- **Unbind when needed**: Prevent memory leaks

Common GUI Mistakes to Avoid

Don't Do This

- Freeze the GUI with long operations
- Create deeply nested layouts
- Use inconsistent styling
- Ignore different screen sizes
- Forget error handling
- Overcomplicate the interface

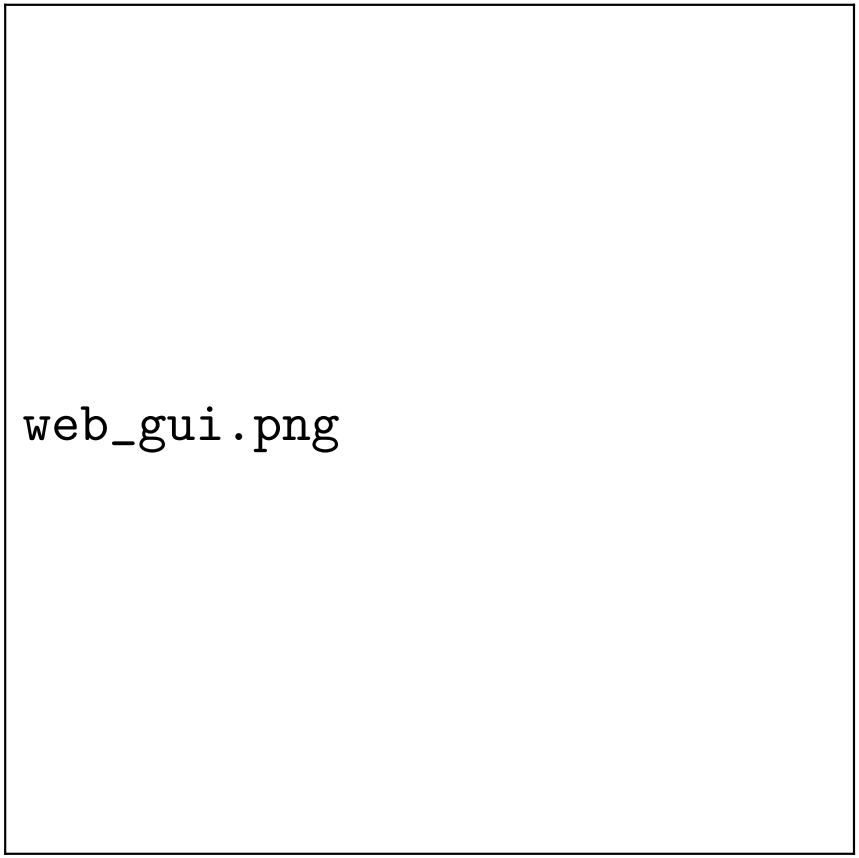
Web-Based GUIs

Advantages

- Cross-platform
- No installation needed
- Easy updates
- Rich ecosystem

Technologies

- HTML/CSS/JavaScript
- React, Vue, Angular
- Electron
- WebAssembly



Mobile GUI Considerations

Touch Interface

- Larger touch targets
- Gesture support
- No hover events

Mobile Constraints

- Smaller screens
- Variable connectivity
- Battery life

Mobile Patterns

- Bottom navigation
- Swipe gestures
- Pull-to-refresh
- Hamburger menus

Key Takeaways

- **GUIs** make software user-friendly and accessible
- **Event-driven programming** responds to user actions
- **Proper layout** and design are crucial for good UX
- **Data display** should be clear and informative
- **Threading** prevents GUI freezing
- **Modern trends** include web and mobile interfaces

Next Steps

Practice Projects

- Create a simple text editor
- Build a file manager
- Make a digital clock with alarms
- Create a drawing application
- Build a simple game (tic-tac-toe)

Learning Resources

- Tkinter documentation
- GUI design pattern books
- Online tutorials and courses
- Open source GUI projects

Questions?

Thank you!