4 Themes

1

# Styling with Themes

In the previous chapters of our Tooltip series, we created a basic ToolTip class, and also created our first Graphical User Interface (GUI). Using starter.py. The graphical user interface, also known as the GUI, is a visual way of interacting with a computer, using things like windows, icons, and menus. This is used by most of all modern operating systems, and apps today.

So, for now we are taking, the GUI, and talking with it to bring about this concept of theming—giving our tooltips personality, mood, and visual resonance.

# Why Themes Matter

Tooltips are more than floating labels—they’re whispers of guidance. A well-themed tooltip can:

1. Enhance readability, and accessibility
2. Match the aesthetic of your application
3. Create emotional tone (e.g., whimsical, professional, mysterious)

Did You Know?  
Tkinter doesn’t include a built-in **ToolTip** widget. What you’re seeing is a custom class that uses a floating **Toplevel** window and a **Label** to simulate tooltip behavior. This means that when you created the ToolTip class, you’re not just using a widget—you’ve inventing one.

2

# Create the Theme

Here’s the theme function from our themes.py:

def moonlight(tooltip):

    """Apply moonlight theme to tooltip."""

    if hasattr(tooltip, 'label') and tooltip.label:

        tooltip.label.config(background="#2c2c54", foreground="#e0e0e0")

The **label** is the **attribute** that the hasattr (this method name, means **has attribute**). The term tooltip here is just a container, it could have been named anything —tip, glow, box\_of\_wisdom. It just needs to be instantiated correctly by the function call.

my\_tooltip = ToolTip(sample\_label, text=sparkle("Welcome, traveler!"))

The actual name for the tooltip widget will be passed in when the function is called, to **apply\_theme**.

apply\_theme(my\_tooltip, theme="moonlight")

# Apply the Theme

You can name your tool tip any thing you want. Here it is named **my\_tooltip**. The **ToolTip**, is the class. So, here a little **‘self’** of class is talking to the **ToolTip** class telling it where it wants to go (sample\_label) and what it wants to say, all entangled in its little parenthesis. Then it throws itself into an instance named **my\_tooltip.** So, now when we use tooltip. It knows where to go, and what to say.

my\_tooltip = ToolTip(sample\_label, text=sparkle("Welcome, traveler!"))

apply\_theme(my\_tooltip, theme="moonlight")

# The code

We are adding on to our files that we already have. So, if you have not created the **starter.py** file, or the **core.py** file, which contains our class please go back to our last few tutorials, and create them.

## themes.py

from .core import ToolTip

def moonlight(tooltip):

    """Apply moonlight theme to tooltip."""

    if hasattr(tooltip, 'label') and tooltip.label:

        tooltip.label.config(background="#2c2c54", foreground="#e0e0e0")

def apply\_theme(tooltip, theme="default"):

    """Apply a theme to the tooltip."""

    if theme == "moonlight":

        moonlight(tooltip)

## starter.py

# starter.py

# Your magical toolkit for tooltip adventures

from tooltip\_magic import sparkle

from tooltip\_magic.themes import apply\_theme

# Optional: preload a sample tooltip object

from tooltip\_magic.core import ToolTip

# Create a sample widget and tooltip (if you're using Tkinter)

import tkinter as tk

root = tk.Tk()

root.title("Tooltip Magic Starter")

root.geometry("300x200")

sample\_label = tk.Label(root, text="Hover me")

sample\_label.pack()

my\_tooltip = ToolTip(sample\_label, text=sparkle("Welcome, traveler!"))

apply\_theme(my\_tooltip, theme="moonlight")

# Ready to run your enchanted GUI

root.mainloop()