Tkinter

⚠	ℹ	↺	TRY IT
Error/ Warning	Information	Flashback	Class Exercise

AGENDA

1. Introduction to Tkinter
2. Tkinter - Making the first window
3. Introduction to Tkinter - Titles and Icons
4. Layouts in Tkinter - Introduction to Layouts
5. Introduction to Widgets in layouts, Grid layout
6. Integration using GUI-Designing User Interface
7. Events
8. Functions and Layouts

ICTACADEMY

1. Introduction to Tkinter

Introduction to Tkinter

Tkinter

- Tkinter is the standard GUI (Graphical User Interface) library for Python, providing a fast and easy way to create GUI applications.

- It is a thin object-oriented layer on top of the Tcl/Tk GUI toolkit.

- The name Tkinter comes from ***Tk inter***face.

Introduction to Tkinter

Features

- Standard library integration
- Cross-platform compatibility
- Ease of use
- Rich widget set
- Event-driven programming
- Flexible layout management
- Customizable appearance
- Canvas widget
- Extensible
- Good documentation
- Integration with other libraries
- Stable

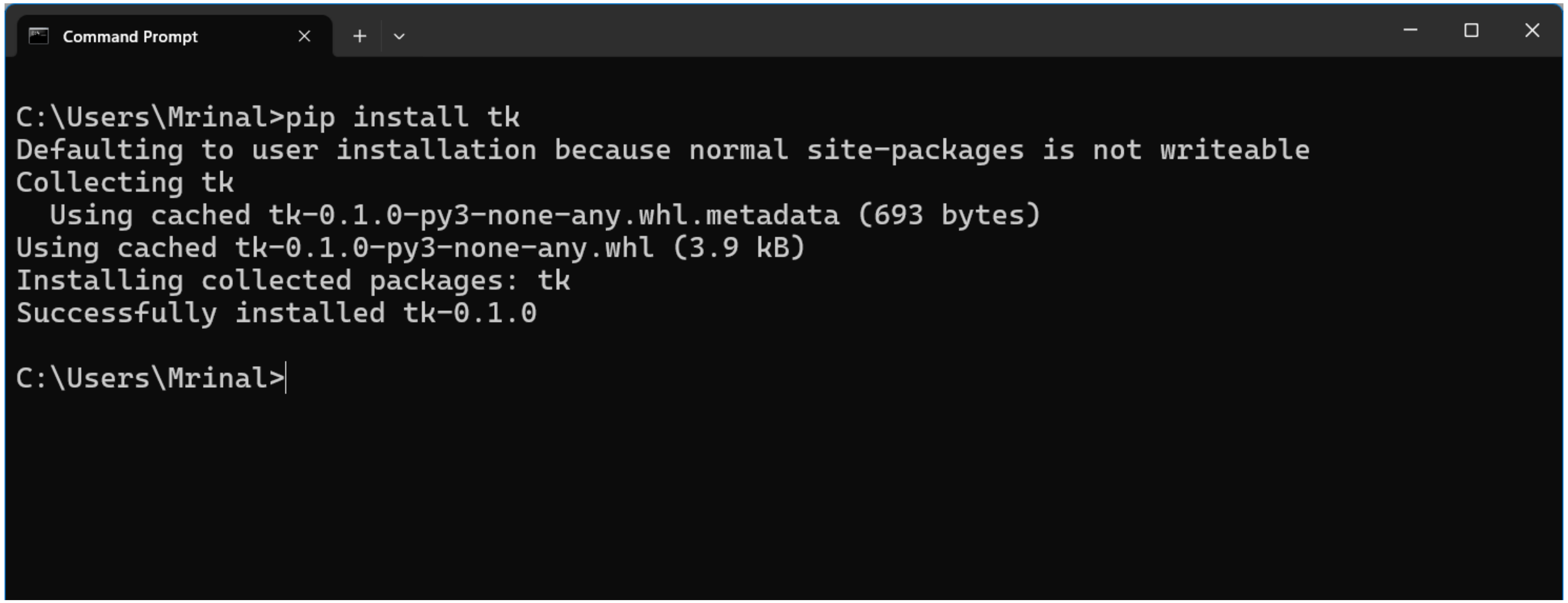
2. Tkinter - Making The First Window

Tkinter – Making the first window

Installing Tkinter

Use `pip` on command prompt to install Tkinter on windows.

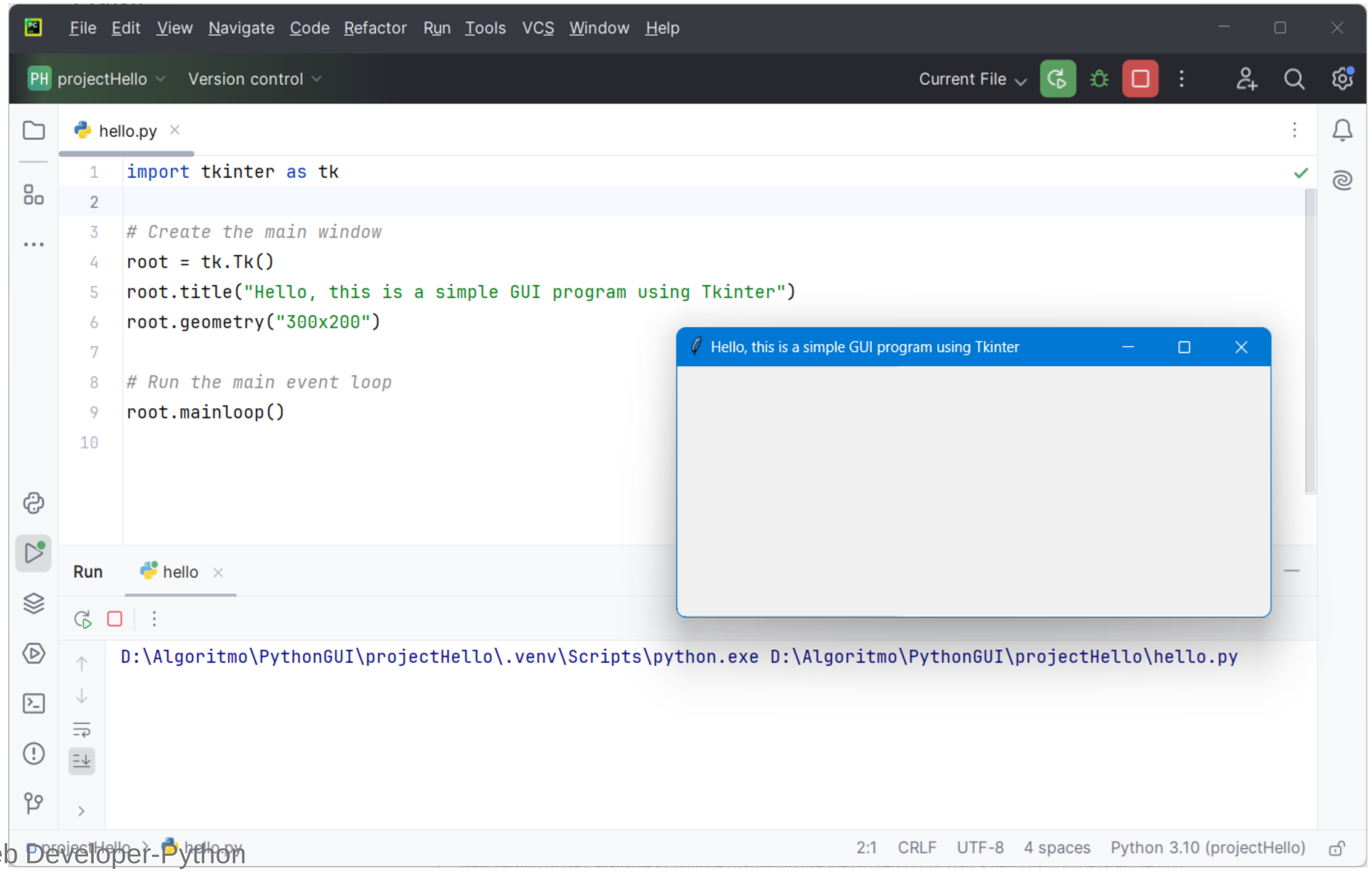
pip install tk



```
C:\Users\Mrinal>pip install tk
Defaulting to user installation because normal site-packages is not writeable
Collecting tk
  Using cached tk-0.1.0-py3-none-any.whl.metadata (693 bytes)
Using cached tk-0.1.0-py3-none-any.whl (3.9 kB)
Installing collected packages: tk
Successfully installed tk-0.1.0

C:\Users\Mrinal>
```

Tkinter – Making the first window

- creating a simple window:


File Edit View Navigate Code Refactor Run Tools VCS Window Help
projectHello
Version control
Current File
hello.py
Run
hello
Hello, this is a simple GUI program using Tkinter
projectHello > hello.py
2:1 CRLF UTF-8 4 spaces Python 3.10 (projectHello)

```python
import tkinter as tk

# Create the main window
root = tk.Tk()
root.title("Hello, this is a simple GUI program using Tkinter")
root.geometry("300x200")

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

```
D:\Algoritmo\PythonGUI\projectHello\.venv\Scripts\python.exe D:\Algoritmo\PythonGUI\projectHello\hello.py
```

Tkinter – Making the first window

Explanation - creating a simple window

```
tk.Tk()
```

Initializes the main window.

```
root.title("Simple GUI")
```

Sets the window title.

```
root.geometry("300x200")
```

Sets the window size.

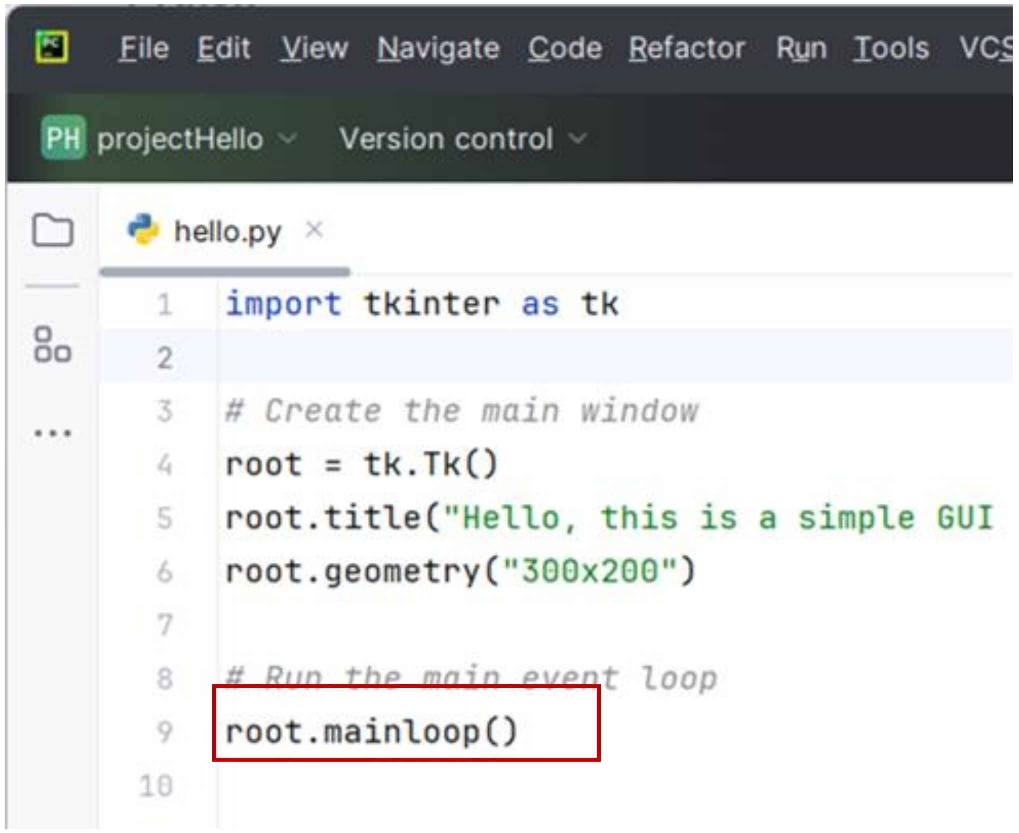
```
root.mainloop()
```

Starts the event loop to keep the window open.

Tkinter – Making the first window

Explanation - creating a simple window

In Tkinter the **root.mainloop()** is a crucial function that starts the main event loop of the GUI application. This loop is responsible for handling all the events and updates within the application.



```
import tkinter as tk

# Create the main window
root = tk.Tk()
root.title("Hello, this is a simple GUI
root.geometry("300x200")

# Run the main event loop
root.mainloop()

```

Tkinter – Making the first window

How it works?

Waiting for Events

Event loop waits for events like mouse click, key press and window action.

Handling Events

When an event occurs, it dispatches the event to the appropriate widget or handler function (e.g. - a button click executes its command).

Continuous Operation

The loop runs continuously until the application window is closed, keeping the GUI active and responsive.

3. Introduction to Tkinter - Titles and Icons

Introduction to Tkinter – Titles and Icons

root.title()

- To set the title of a Tkinter window, use the title() method of the Tkinter window object.
- The title() method takes a single argument, which is a string representing the title you want to display.

```
root.title("My Tkinter App")
```

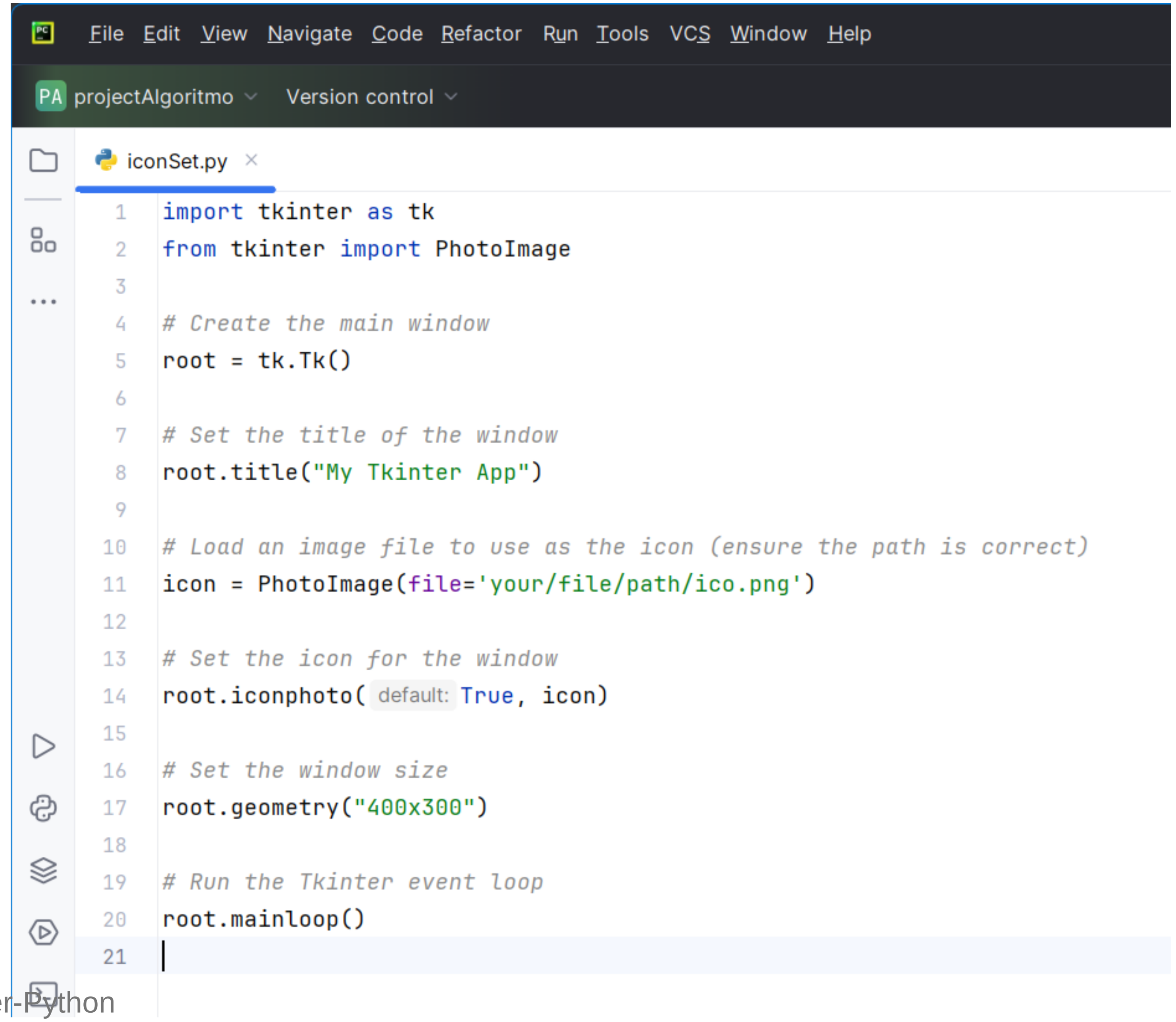
root.iconphoto()

- To set the window icon, use the iconphoto() method of the Tkinter window object.
- This method takes two arguments: a Boolean value and a PhotoImage object representing the icon.

```
icon = PhotoImage(file='path/to/icon.png')
root.iconphoto(True, icon)
```

Introduction to Tkinter – Titles and Icons

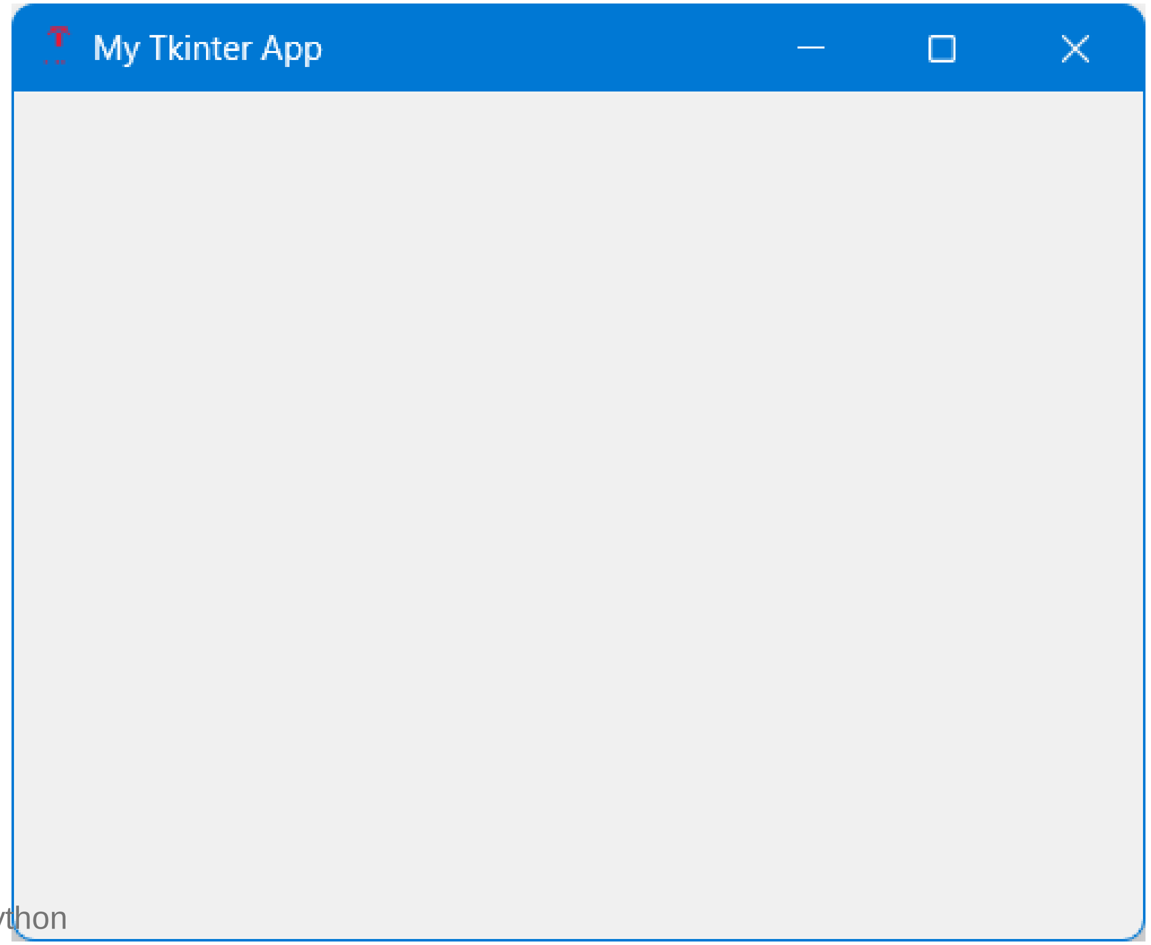
Example


File Edit View Navigate Code Refactor Run Tools VCS Window Help
PA projectAlgoritmo
Version control
iconSet.py

```python
import tkinter as tk
from tkinter import PhotoImage

# Create the main window
root = tk.Tk()

# Set the title of the window
root.title("My Tkinter App")

# Load an image file to use as the icon (ensure the path is correct)
icon = PhotoImage(file='your/file/path/ico.png')

# Set the icon for the window
root.iconphoto( default: True, icon)

# Set the window size
root.geometry("400x300")

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

Introduction to Tkinter – Titles and Icons

- The icon image file should be in a format supported by PhotoImage, such as PNG, GIF, or PPM/PGM.
- Import PhotoImage from the Tkinter library to handle image files.



4. Layouts in Tkinter - Introduction to Layouts

Layouts in Tkinter – Introduction to layouts

Explanation

In Tkinter, a layout refers to the way widgets (such as buttons, labels, text boxes, etc.) are organized within a window or frame.

Type of layouts

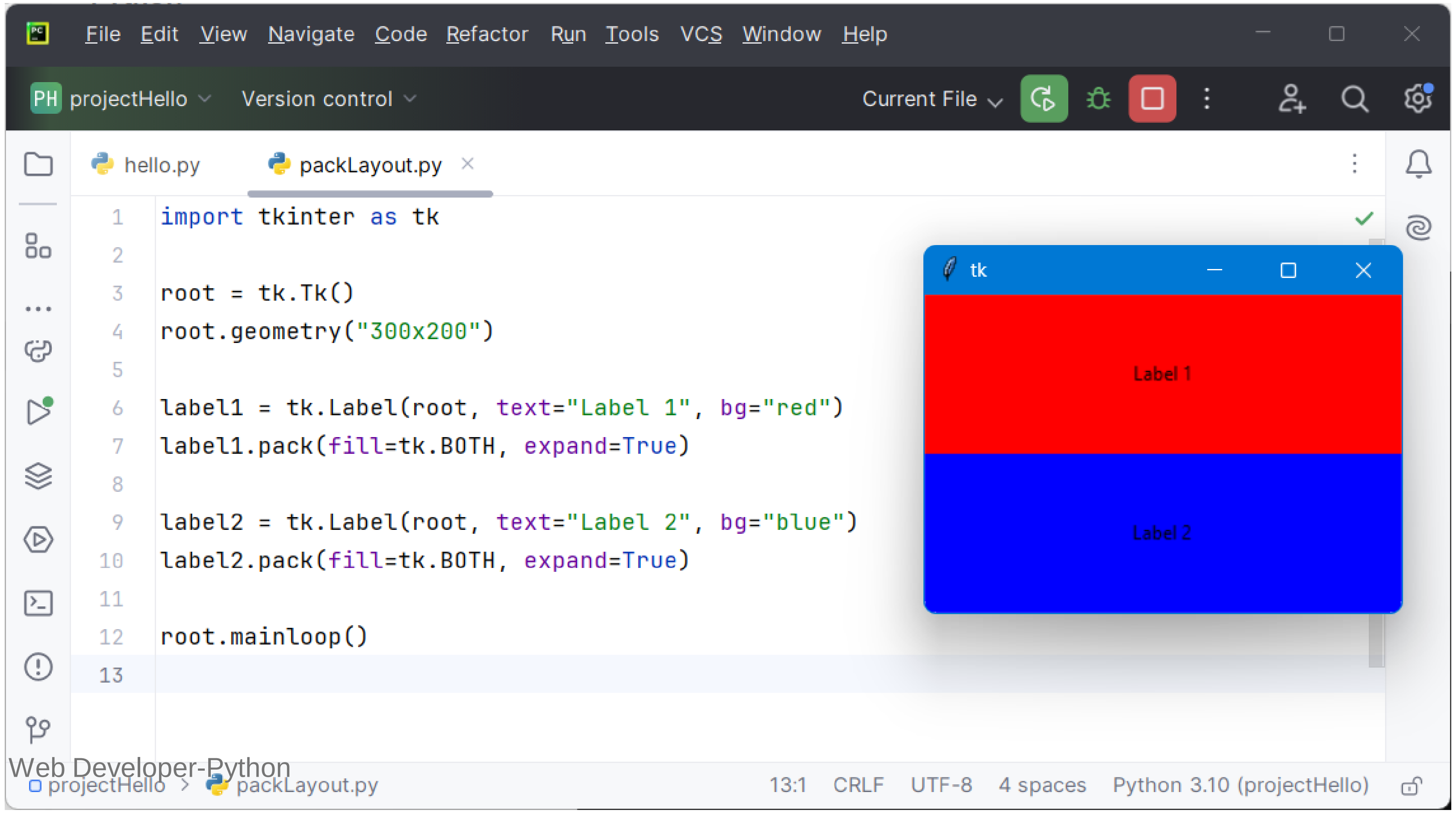
Tkinter offers several ways to place widgets in the window, the most commons are -

- Pack layout
- Grid layout
- Place layout

Layouts in Tkinter – Introduction to layouts

Pack layout

The `pack` geometry manager organizes widgets in blocks before placing them in the parent widget.



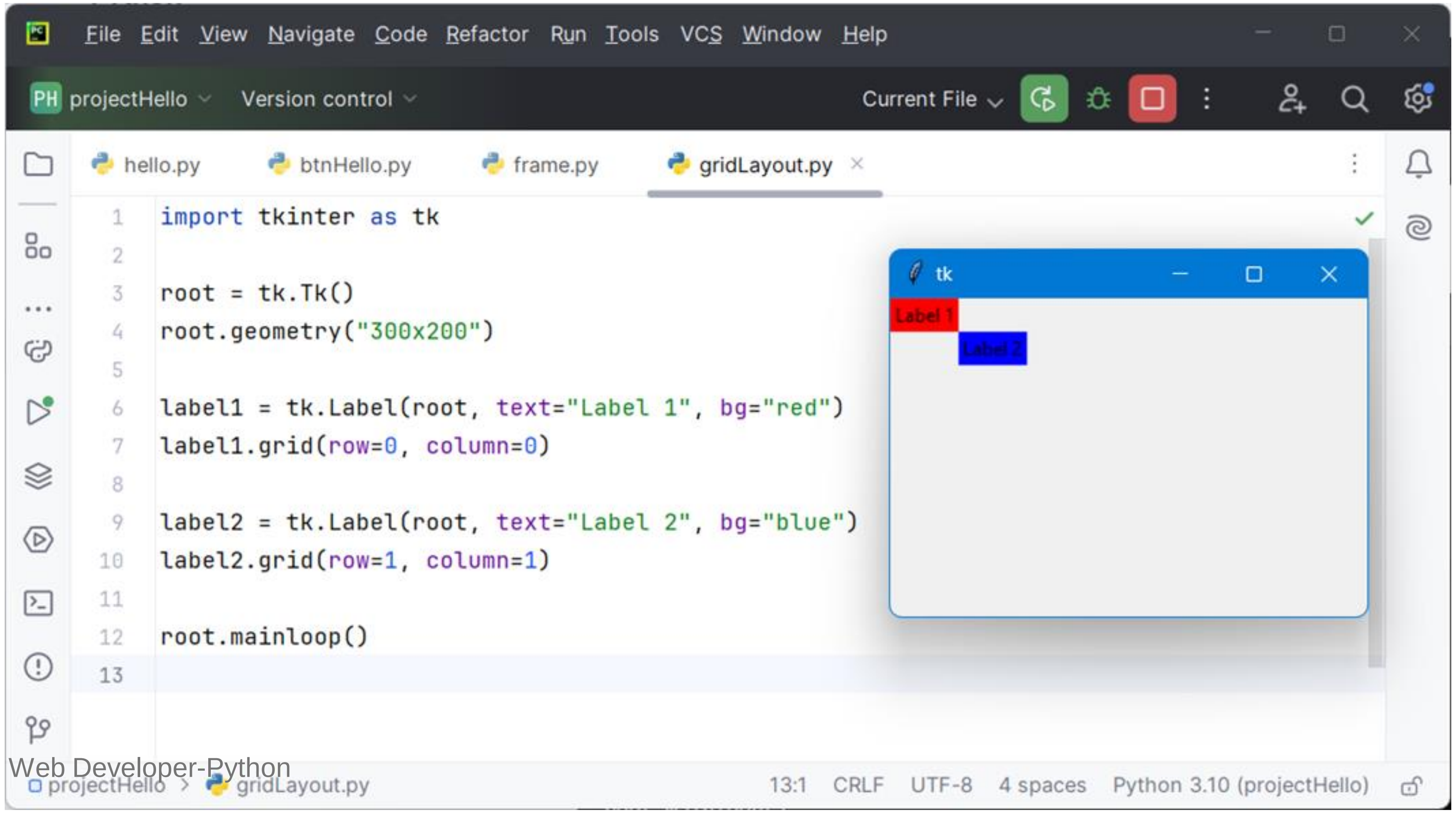
```
import tkinter as tk

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

label1 = tk.Label(root, text="Label 1", bg="red")
label1.pack(fill=tk.BOTH, expand=True)

label2 = tk.Label(root, text="Label 2", bg="blue")
label2.pack(fill=tk.BOTH, expand=True)

root.mainloop()
```

File Edit View Navigate Code Refactor Run Tools VCS Window Help
projectHello
Version control
Current File
hello.py
packLayout.py
tk
Label 1
Label 2
projectHello > packLayout.py
13:1 CRLF UTF-8 4 spaces Python 3.10 (projectHello)

Layouts in Tkinter – Introduction to layouts

Grid layout

The `grid` geometry manager organizes widgets in a table-like structure using rows and columns.

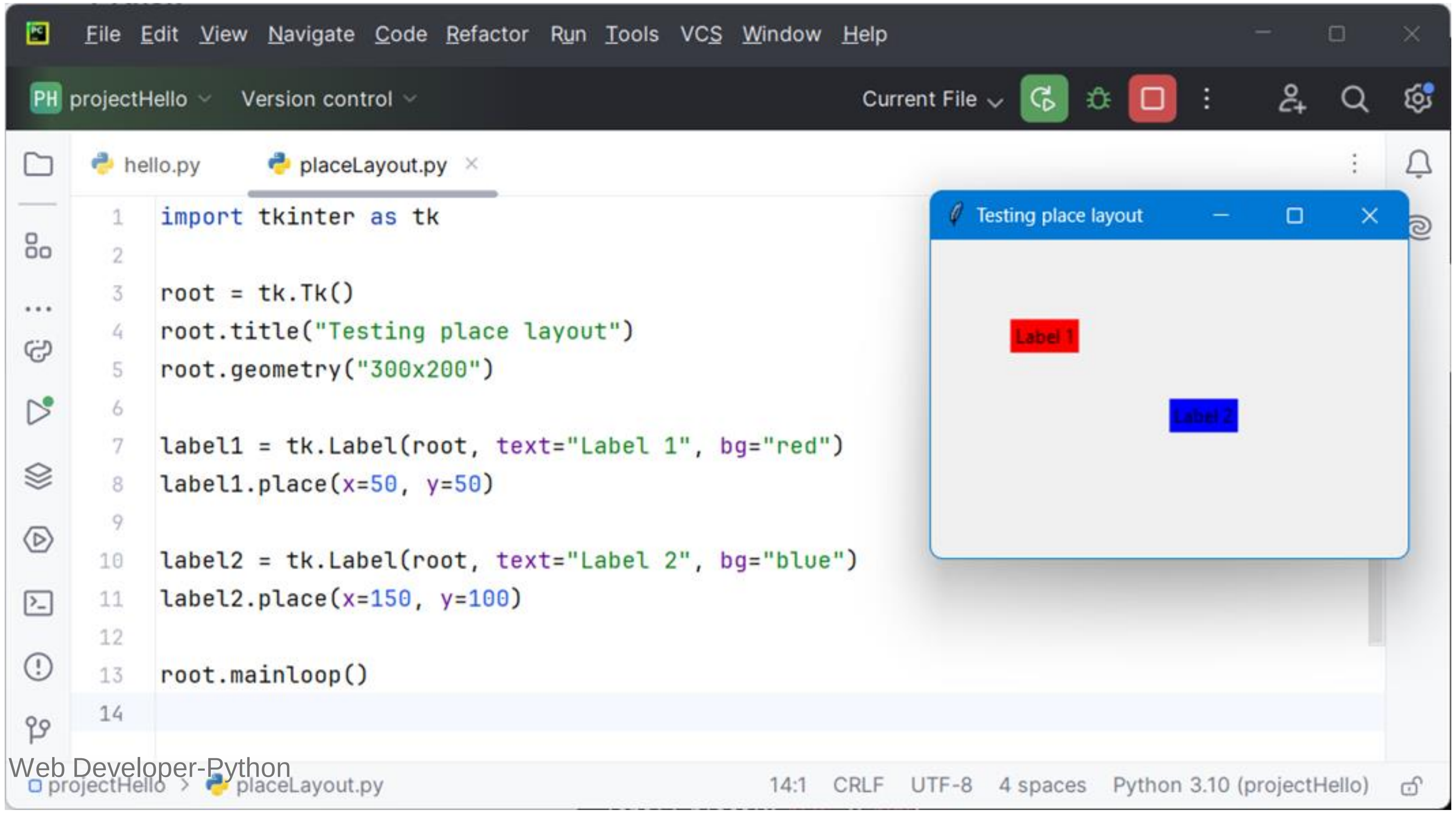

File Edit View Navigate Code Refactor Run Tools VCS Window Help
projectHello
Version control
Current File
hello.py
btnHello.py
frame.py
gridLayout.py
tk
Label 1
Label 2
projectHello > gridLayout.py
13:1 CRLF UTF-8 4 spaces Python 3.10 (projectHello)

```python
import tkinter as tk

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

label1 = tk.Label(root, text="Label 1", bg="red")
label1.grid(row=0, column=0)

label2 = tk.Label(root, text="Label 2", bg="blue")
label2.grid(row=1, column=1)

root.mainloop()
```

Layouts in Tkinter – Introduction to layouts

Place layout

The `place` geometry manager allows you to position widgets at an absolute location within the parent widget.


File Edit View Navigate Code Refactor Run Tools VCS Window Help
projectHello
Version control
Current File
hello.py
placeLayout.py
Testing place layout
Label 1
Label 2
projectHello > placeLayout.py
14:1 CRLF UTF-8 4 spaces Python 3.10 (projectHello)

```python
import tkinter as tk

root = tk.Tk()
root.title("Testing place layout")
root.geometry("300x200")

label1 = tk.Label(root, text="Label 1", bg="red")
label1.place(x=50, y=50)

label2 = tk.Label(root, text="Label 2", bg="blue")
label2.place(x=150, y=100)

root.mainloop()
```

Layouts in Tkinter – Introduction to layouts

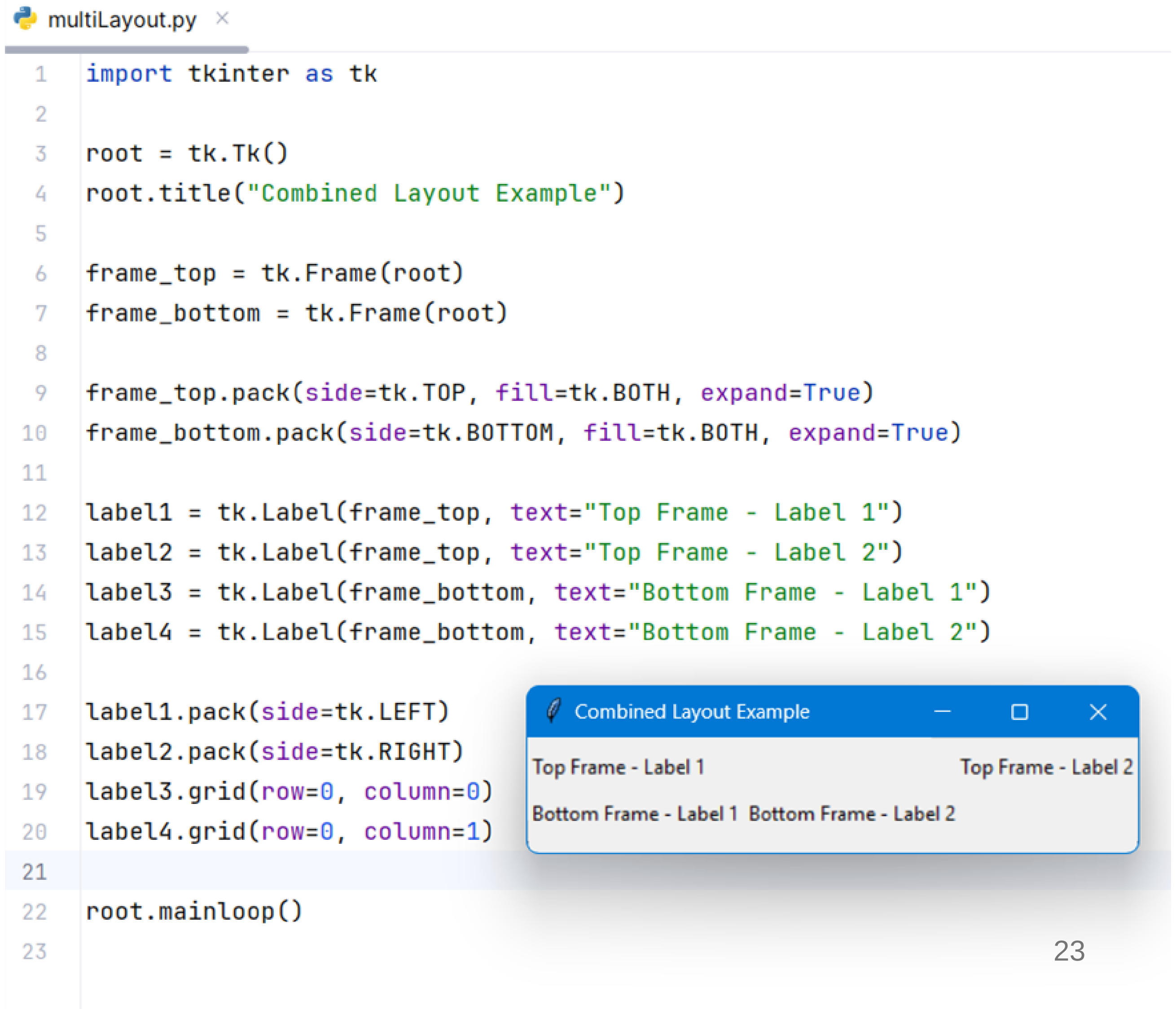
Choosing a layout manager

- **Pack**: Suitable for simple layouts where you want widgets stacked vertically or horizontally.
- **Grid**: Ideal for more complex layouts involving rows and columns, similar to a table.
- **Place**: Useful for precise control over widget positioning for complex interfaces.

Layouts in Tkinter – Introduction to layouts

Combining layout managers

You can also combine layout managers within different containers to create complex layouts. It's common to use a Frame widget to group related widgets and apply different layout strategies within each frame.


multiLayout.py

```python
import tkinter as tk

root = tk.Tk()
root.title("Combined Layout Example")

frame_top = tk.Frame(root)
frame_bottom = tk.Frame(root)

frame_top.pack(side=tk.TOP, fill=tk.BOTH, expand=True)
frame_bottom.pack(side=tk.BOTTOM, fill=tk.BOTH, expand=True)

label1 = tk.Label(frame_top, text="Top Frame - Label 1")
label2 = tk.Label(frame_top, text="Top Frame - Label 2")
label3 = tk.Label(frame_bottom, text="Bottom Frame - Label 1")
label4 = tk.Label(frame_bottom, text="Bottom Frame - Label 2")

label1.pack(side=tk.LEFT)
label2.pack(side=tk.RIGHT)
label3.grid(row=0, column=0)
label4.grid(row=0, column=1)

root.mainloop()
```

```
Combined Layout Example
Top Frame - Label 1                         Top Frame - Label 2
Bottom Frame - Label 1  Bottom Frame - Label 2
```

Layouts in Tkinter – Introduction to layouts

TRY IT

Create a Tkinter application that displays a window with a title, an icon, and a simple layout using both pack and grid. Make the window size = (400x400).

5. Introduction to Widgets in Layouts, Grid Layout

Introduction to widgets in layouts, grid layout

Widgets

Widgets are the building blocks of a Tkinter GUI application.

Here, example of some widgets provided by Tkinter -

1. Label
2. Button
3. Entry
4. Text
5. Checkbutton
6. Radiobutton
7. Listbox
8. Combobox
9. Scale
10. Spinbox
11. Scrollbar
12. Frame
13. Canvas
14. Menu

Introduction to widgets in layouts, grid layout

Widgets in layouts

In Tkinter, we use geometry managers to control the layout of widgets within a window or a container (like a frame). The three main geometry managers are pack, grid, and place. Among these, the grid geometry manager is highly flexible and allows for more complex and precise widget arrangements.

Grid layouts

The grid geometry manager organizes widgets in a table-like structure, defined by rows and columns. This approach is similar to how a spreadsheet or HTML table works.

Introduction to widgets in layouts, grid layout

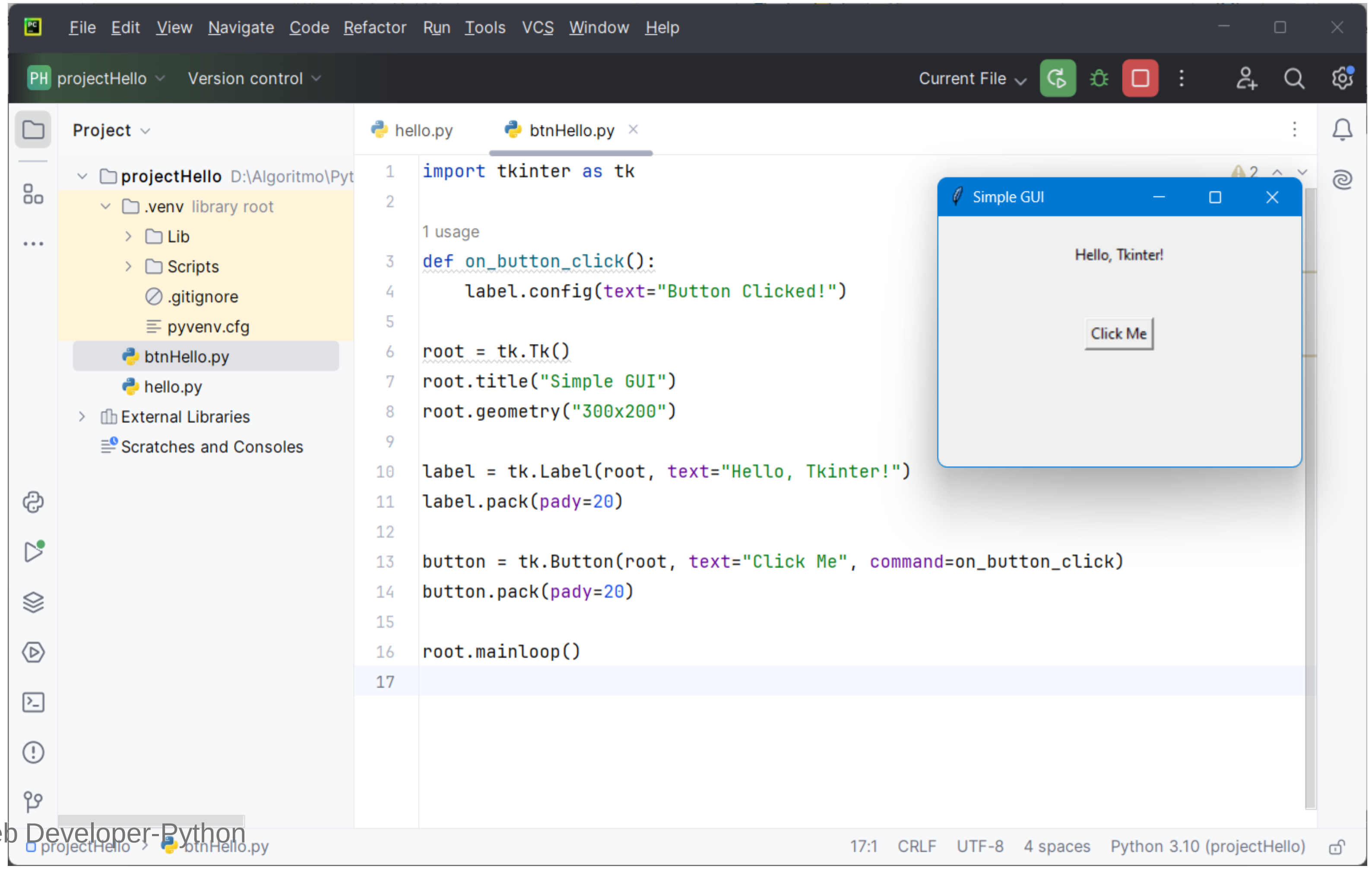
Parameters of grid()

```
label.grid(row=1, column=0, columnspan=2, rowspan=2, padx=10, pady=20, ipadx=5, ipady=5,
sticky="nsew", in_=container_widget)
```

- row: Specifies the row index of the grid.
- column: Specifies the column index of the grid.
- rowspan: Specifies the number of rows a widget should span.
- columnspan: Specifies the number of columns a widget should span.
- sticky: Specifies which sides of the cell the widget should stick to (n, s, e, w, or combinations like ns, ew, nsew).
- padx: Specifies horizontal padding around the widget.
- pady: Specifies vertical padding around the widget.
- ipadx: Specifies internal horizontal padding within the widget.
- ipady: Specifies internal vertical padding within the widget.
- in_: Specifies the parent widget for the grid.

Introduction to widgets in layouts, grid layout

Adding label and button to pack layout



```python
import tkinter as tk

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

root = tk.Tk()
root.title("Simple GUI")
root.geometry("300x200")

label = tk.Label(root, text="Hello, Tkinter!")
label.pack(pady=20)

button = tk.Button(root, text="Click Me", command=on_button_click)
button.pack(pady=20)

root.mainloop()
```

Introduction to widgets in layouts, grid layout

explanation - adding label and button to pack layout

tk.Label()
Creates a label widget.

label.pack()
Adds the label to the window.

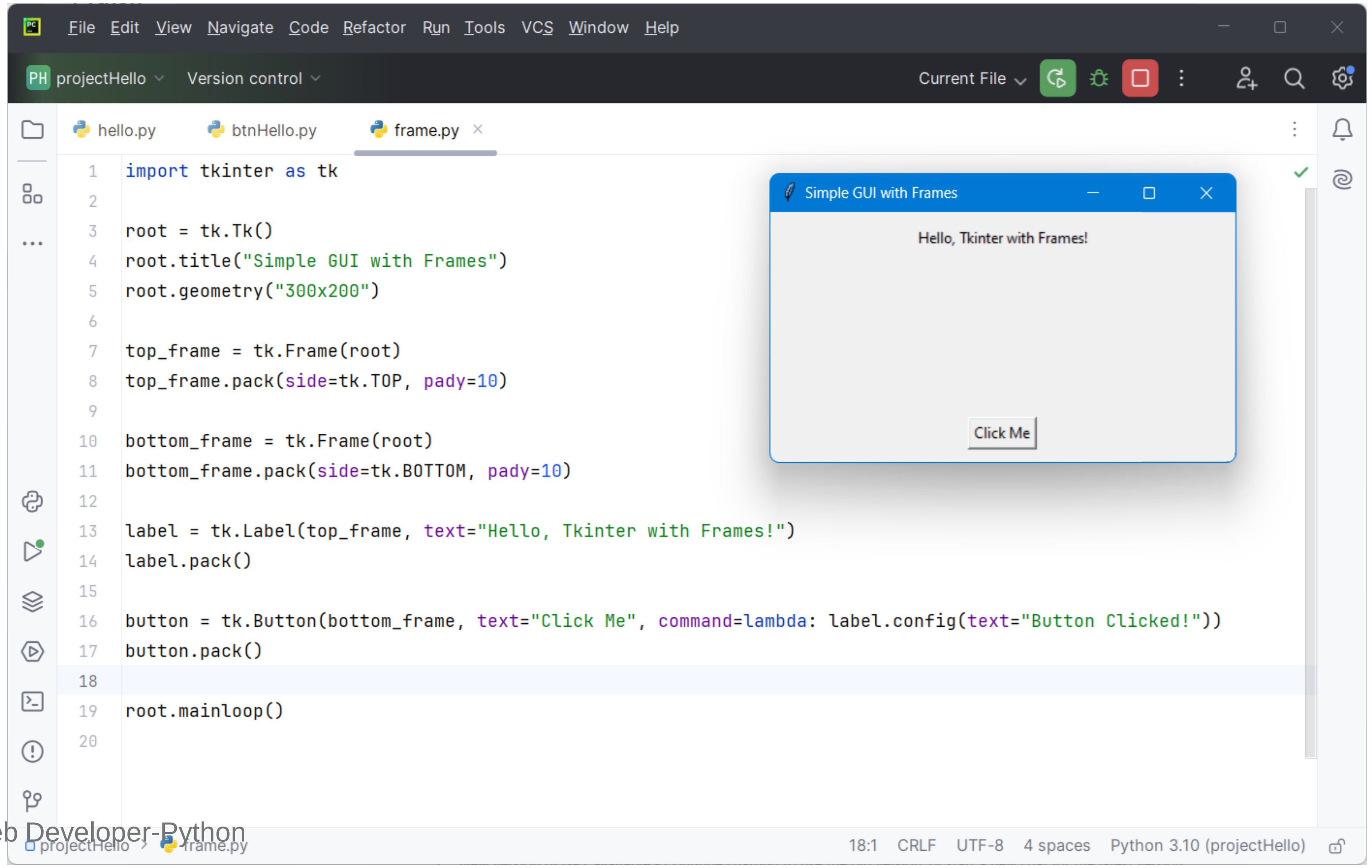
tk.Button()
Creates a button widget.

button.pack()
Adds the button to the window.

command=on_button_click
Specifies the function to call when the button is clicked.

Introduction to widgets in layouts, grid layout

Organizing widgets with frame


File Edit View Navigate Code Refactor Run Tools VCS Window Help
projectHello
Version control
Current File
hello.py
btnHello.py
frame.py
projectHello › frame.py
18:1 CRLF UTF-8 4 spaces Python 3.10 (projectHello)

```python
import tkinter as tk

root = tk.Tk()
root.title("Simple GUI with Frames")
root.geometry("300x200")

top_frame = tk.Frame(root)
top_frame.pack(side=tk.TOP, pady=10)

bottom_frame = tk.Frame(root)
bottom_frame.pack(side=tk.BOTTOM, pady=10)

label = tk.Label(top_frame, text="Hello, Tkinter with Frames!")
label.pack()

button = tk.Button(bottom_frame, text="Click Me", command=lambda: label.config(text="Button Clicked!"))
button.pack()

root.mainloop()
```

Simple GUI with Frames

Hello, Tkinter with Frames!

Click Me

Introduction to widgets in layouts, grid layout

Explanation - organizing widgets with frame

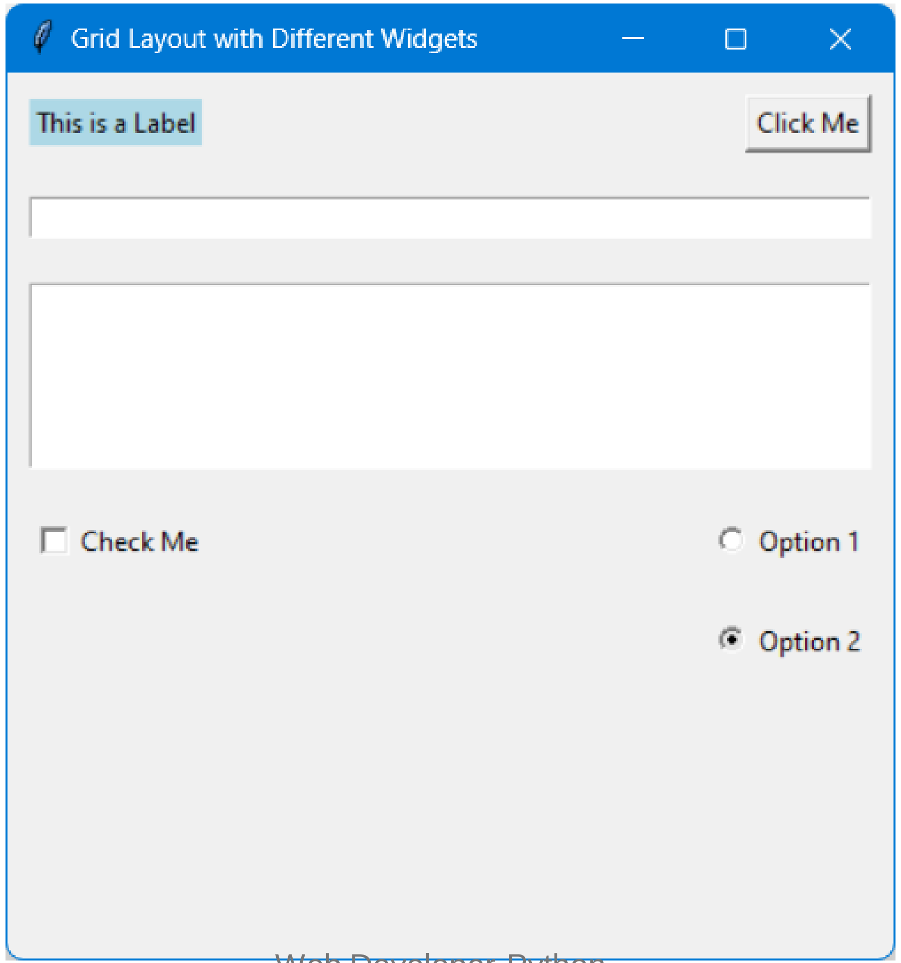
tk.Frame()
Creates a frame to group widgets.

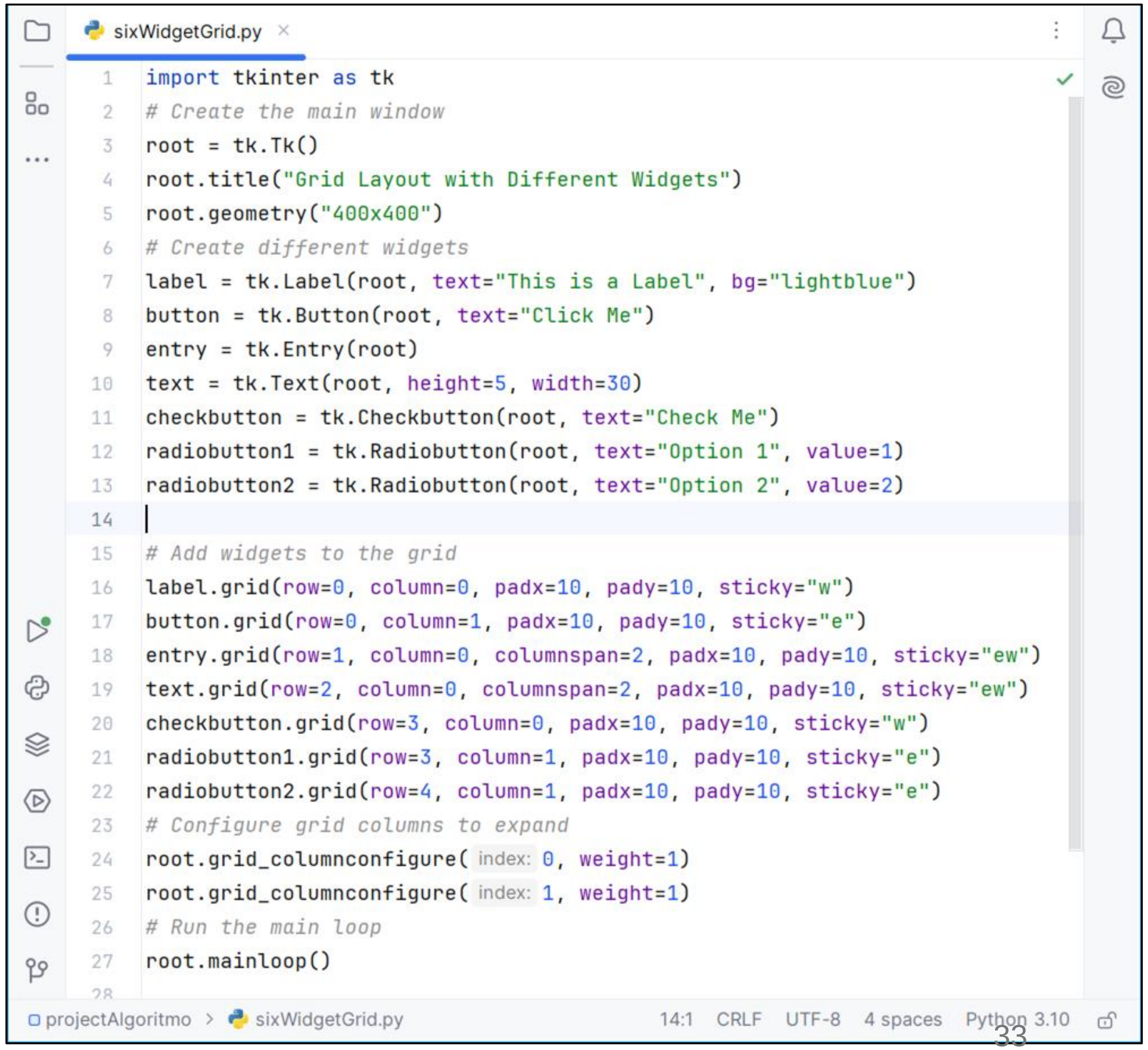
frame.pack()
Adds the frame to the window.

Widgets can be added to specific frames for better organization.

Introduction to widgets in layouts, grid layout

Adding multiple widgets to grid layout




sixWidgetGrid.py

```
import tkinter as tk
# Create the main window
root = tk.Tk()
root.title("Grid Layout with Different Widgets")
root.geometry("400x400")
# Create different widgets
label = tk.Label(root, text="This is a Label", bg="lightblue")
button = tk.Button(root, text="Click Me")
entry = tk.Entry(root)
text = tk.Text(root, height=5, width=30)
checkbutton = tk.Checkbutton(root, text="Check Me")
radiobutton1 = tk.Radiobutton(root, text="Option 1", value=1)
radiobutton2 = tk.Radiobutton(root, text="Option 2", value=2)

# Add widgets to the grid
label.grid(row=0, column=0, padx=10, pady=10, sticky="w")
button.grid(row=0, column=1, padx=10, pady=10, sticky="e")
entry.grid(row=1, column=0, columnspan=2, padx=10, pady=10, sticky="ew")
text.grid(row=2, column=0, columnspan=2, padx=10, pady=10, sticky="ew")
checkbutton.grid(row=3, column=0, padx=10, pady=10, sticky="w")
radiobutton1.grid(row=3, column=1, padx=10, pady=10, sticky="e")
radiobutton2.grid(row=4, column=1, padx=10, pady=10, sticky="e")
# Configure grid columns to expand
root.grid_columnconfigure( index: 0, weight=1)
root.grid_columnconfigure( index: 1, weight=1)
# Run the main loop
root.mainloop()
```

projectAlgoritmo > sixWidgetGrid.py
14:1 CRLF UTF-8 4 spaces Python 3.10

Introduction to widgets in layouts, grid layout

Explanation - adding multiple widgets to grid layout

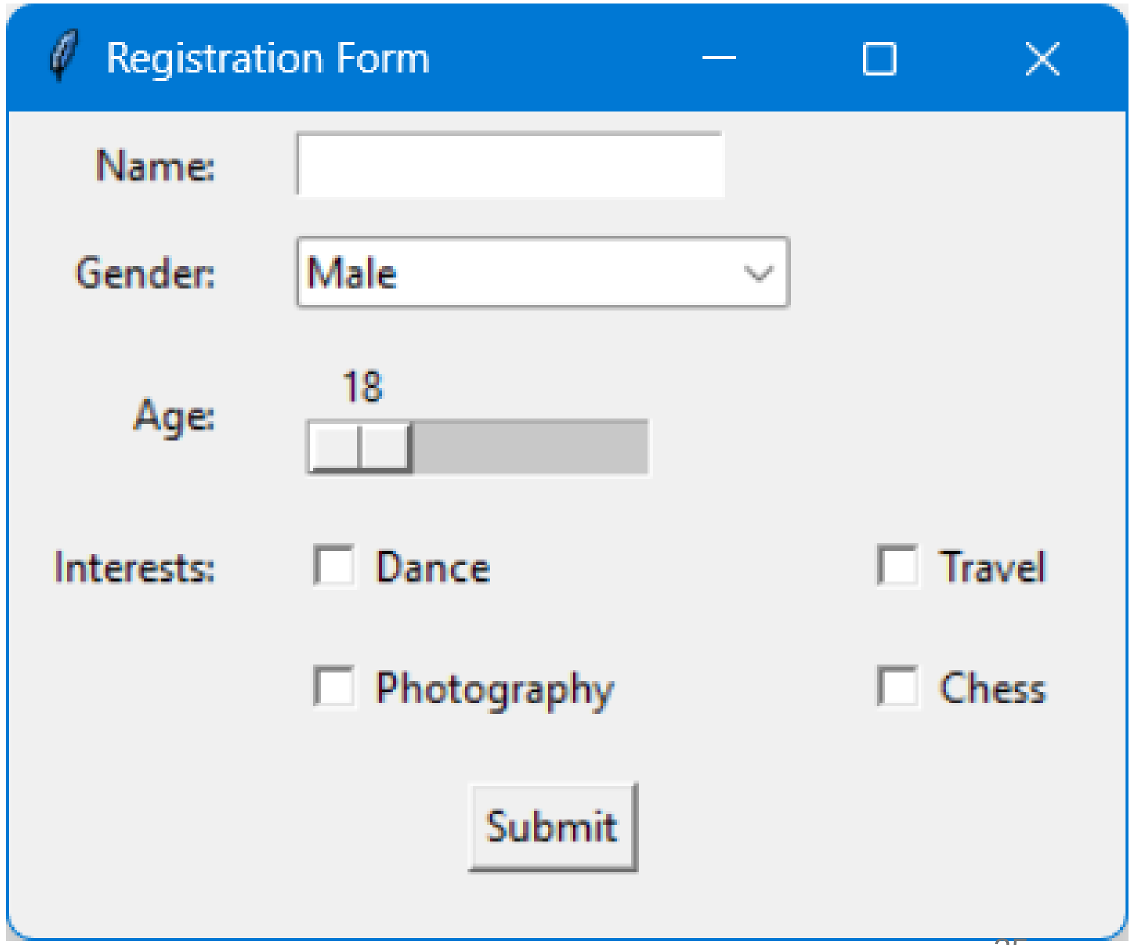
Each widget is placed in the grid using the grid method with specified row, column, padding, and sticky options.

- label is placed at row 0, column 0.
- button is placed at row 0, column 1.
- entry is placed at row 1, column 0, spanning 2 columns.
- text is placed at row 2, column 0, spanning 2 columns.
- checkbutton is placed at row 3, column 0.
- radiobutton1 is placed at row 3, column 1.
- radiobutton2 is placed at row 4, column 1.

GUI Development – Introduction

TRY IT

Write python code using Tkinter library to design the registration form. Use grid layout.

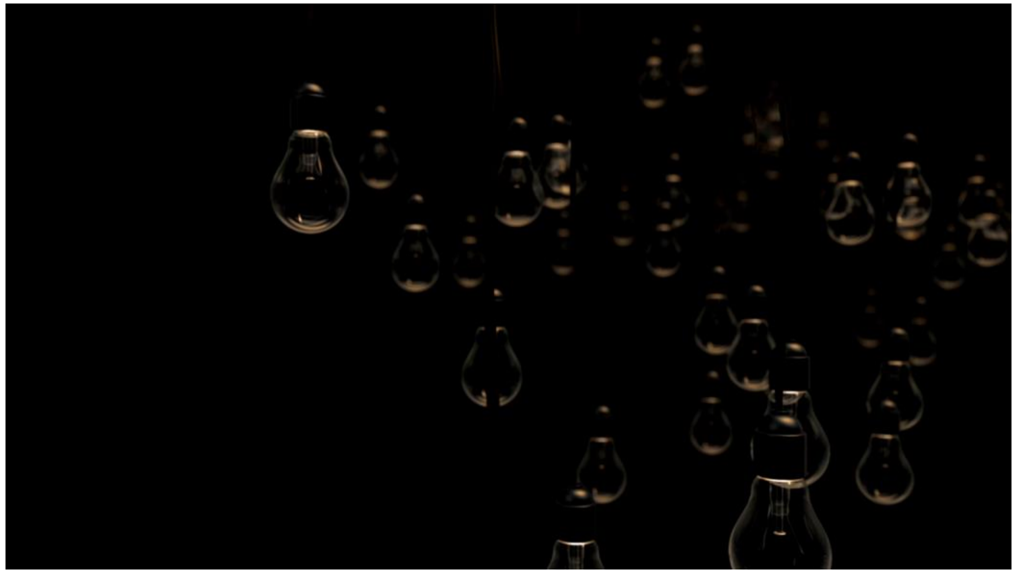


6. Integration Using GUI-Designing User Interface

Integration using GUI–designing user interface

Explanation

Integrating GUI design involves creating a visually appealing and functional interface using a GUI toolkit, connecting UI elements to program logic, handling user input and output, and ensuring smooth interaction between the UI and back-end functionality.

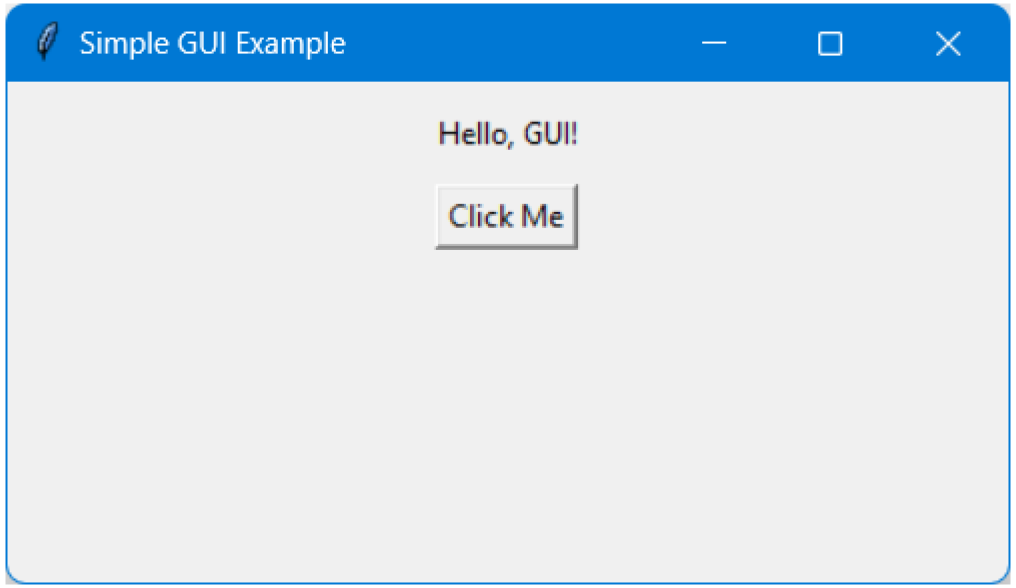


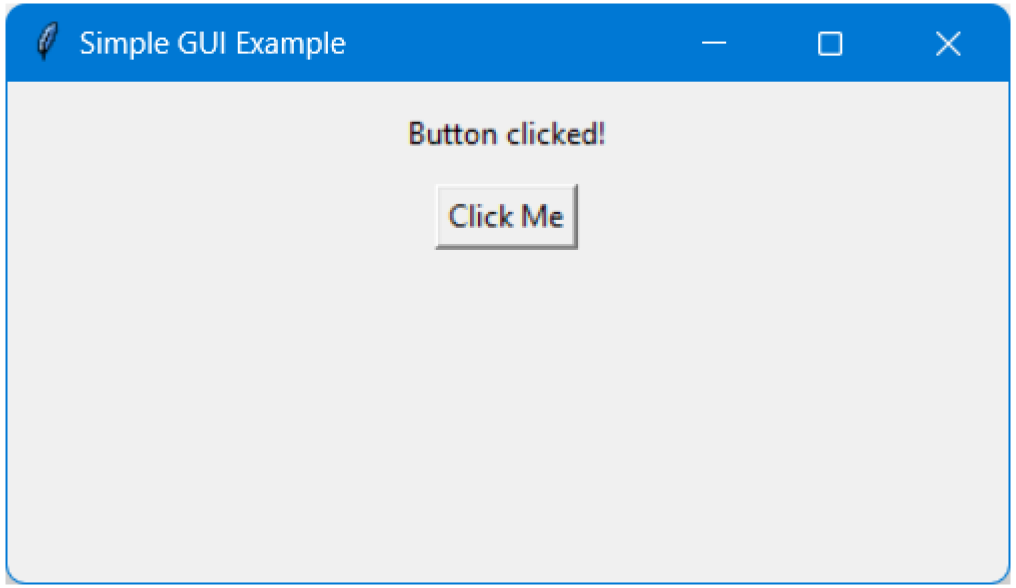
In the video - whenever the bigger bulb is appearing it turns on and turns off before disappearing.

Here, GUI (i.e. the bulb) is connected with the program logic to turn on/off considering event/situation.

Integration using GUI-designing user interface

Example – GUI of “Click Me”



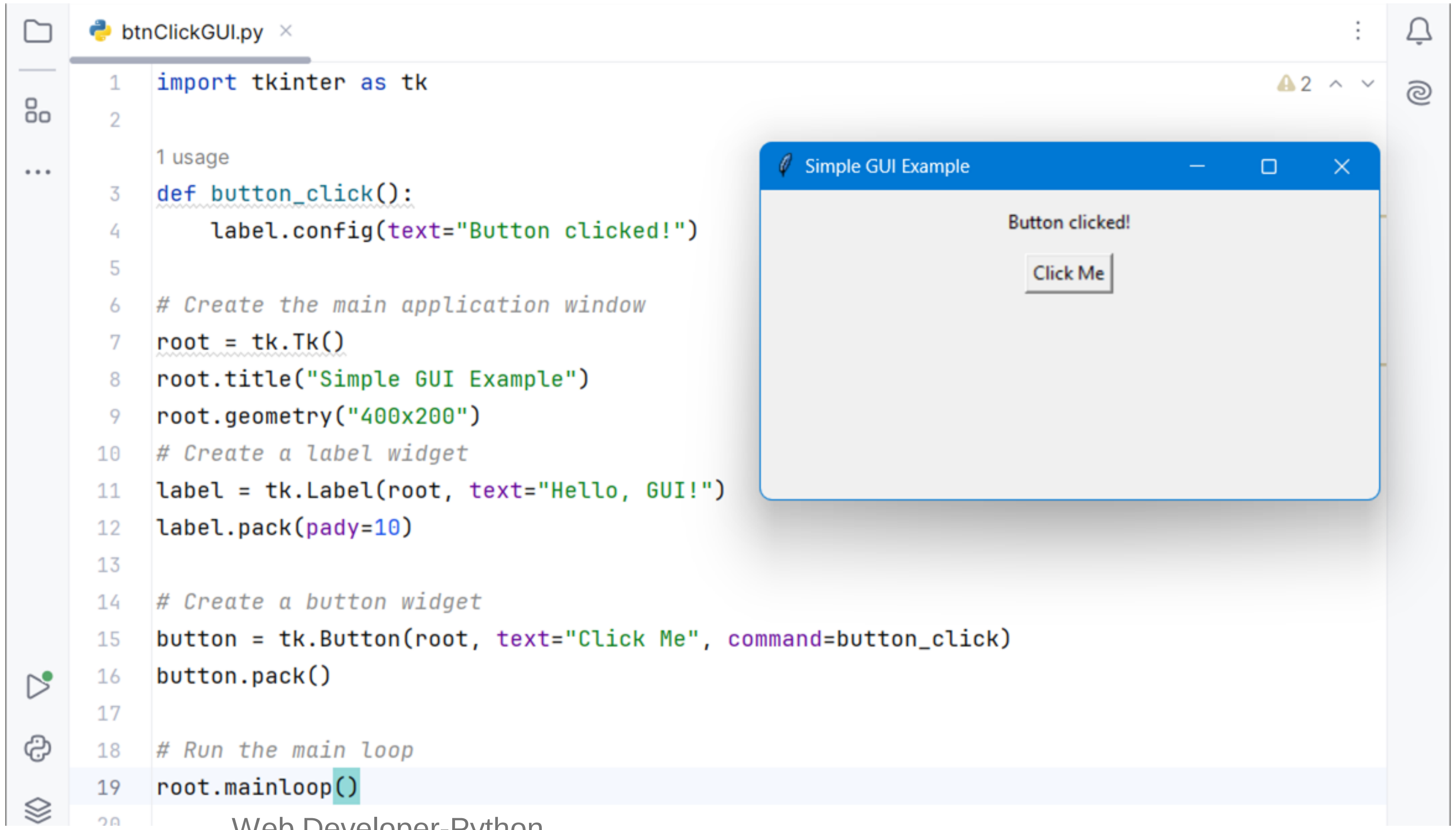


In this example, tkinter is used to create a simple window (root). A label (label) and a button (button) are added to root. Clicking the button triggers the button_click function, which updates the label's text from “Hello, GUI!” to “Button clicked”.

Integration using GUI-designing user interface

Example – Code of “Click Me”

Here is code for that GUI integration with program logic to change the label text on button click.



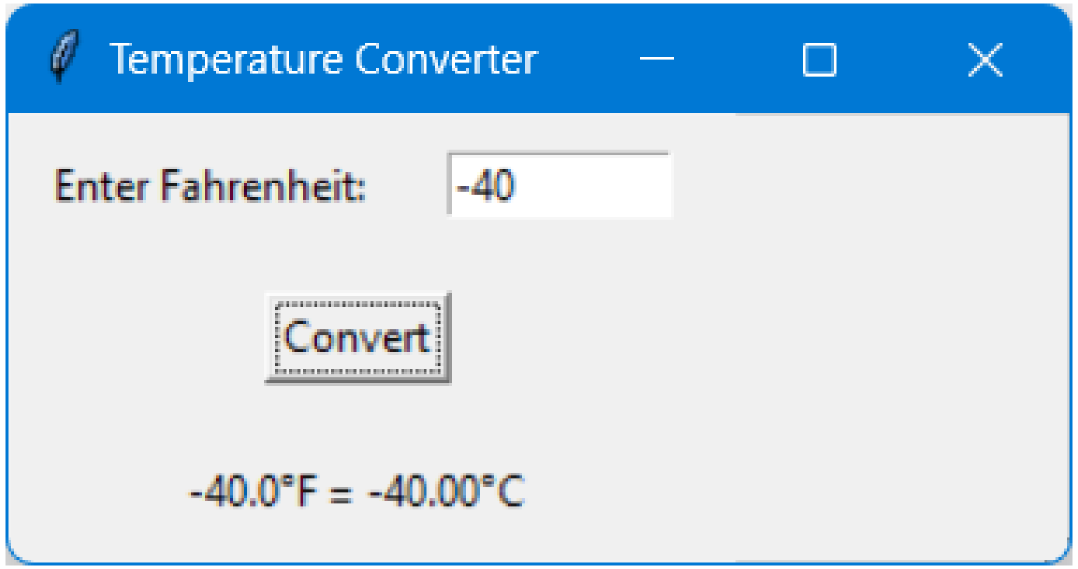
```python
import tkinter as tk

1 usage
def button_click():
    label.config(text="Button clicked!")

# Create the main application window
root = tk.Tk()
root.title("Simple GUI Example")
root.geometry("400x200")
# Create a label widget
label = tk.Label(root, text="Hello, GUI!")
label.pack(pady=10)

# Create a button widget
button = tk.Button(root, text="Click Me", command=button_click)
button.pack()

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

```
Button clicked!
Click Me
```

Integration using GUI-designing user interface

Example – GUI of temperature converter

This example creates a simple Fahrenheit to Celsius temperature converter. It uses an entry widget (entry) for user input, a button (convert_button) to trigger the conversion function (convert), and a label (result_label) to display the conversion result.



```
-40.0°F = -40.00°C
```

Integration using GUI-designing user interface

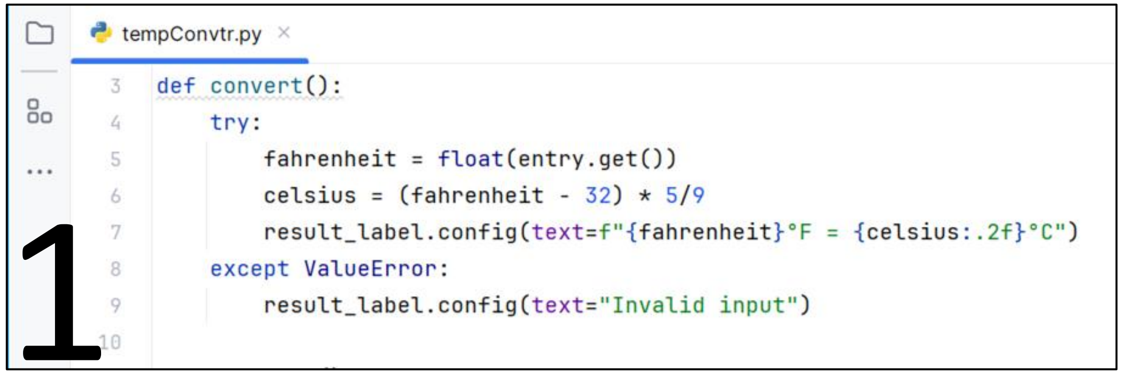
Question : How do we handle run time error in Python?

Answer : Using try-except-else-finally block.

Integration using GUI-designing user interface

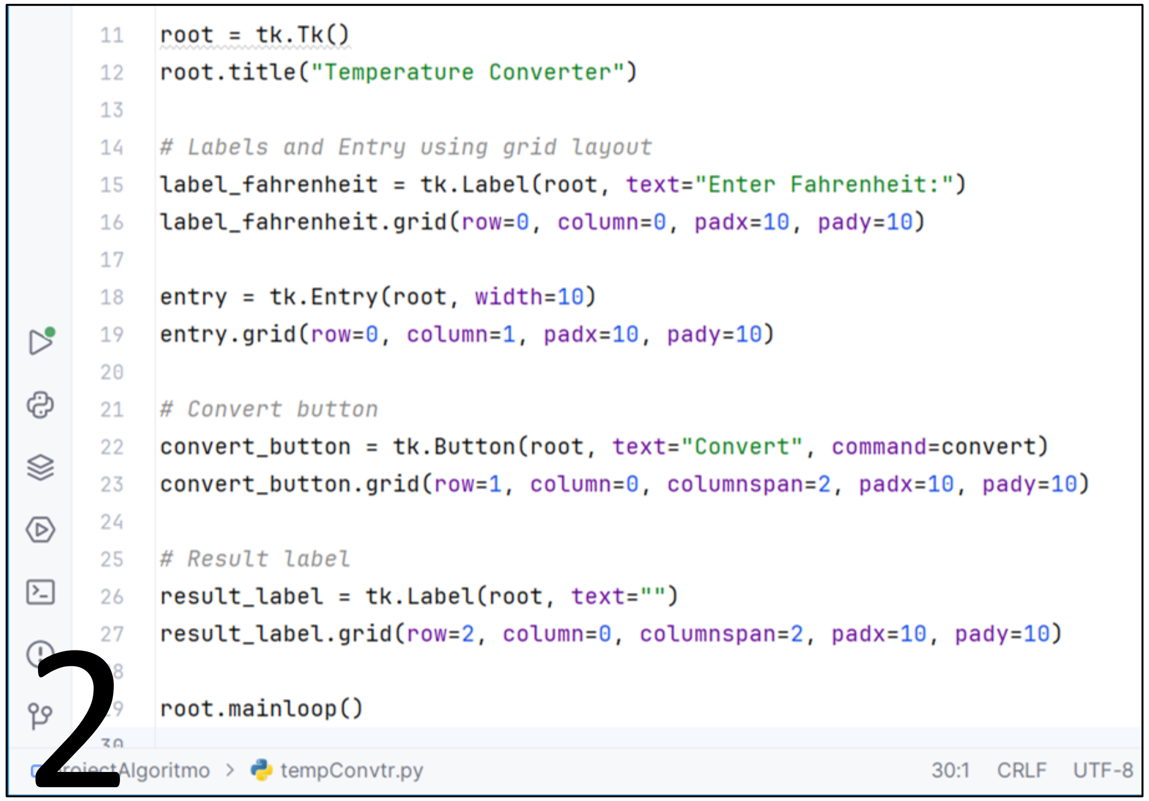
Example – Code of temperature converter

1. Defining the convert() function to convert the temperature.
2. Designing the GUI of converter and connecting with the convert() function.



1

```python
def convert():
    try:
        fahrenheit = float(entry.get())
        celsius = (fahrenheit - 32) * 5/9
        result_label.config(text=f"{fahrenheit}°F = {celsius:.2f}°C")
    except ValueError:
        result_label.config(text="Invalid input")
```



2

```python
root = tk.Tk()
root.title("Temperature Converter")

# Labels and Entry using grid layout
label_fahrenheit = tk.Label(root, text="Enter Fahrenheit:")
label_fahrenheit.grid(row=0, column=0, padx=10, pady=10)

entry = tk.Entry(root, width=10)
entry.grid(row=0, column=1, padx=10, pady=10)

# Convert button
convert_button = tk.Button(root, text="Convert", command=convert)
convert_button.grid(row=1, column=0, columnspan=2, padx=10, pady=10)

# Result label
result_label = tk.Label(root, text="")
result_label.grid(row=2, column=0, columnspan=2, padx=10, pady=10)

root.mainloop()
```

Integration using GUI-designing user interface

Explanation –Temperature converter

- Functionality:

The convert() function is called when the "Convert" button is clicked. It retrieves the Fahrenheit temperature from the entry widget, performs the conversion to Celsius, and updates the result_label with the converted temperature.

- Grid Layout Control:

“padx” and “pady” parameters add padding around widgets to improve spacing and appearance.

7. Events

Events

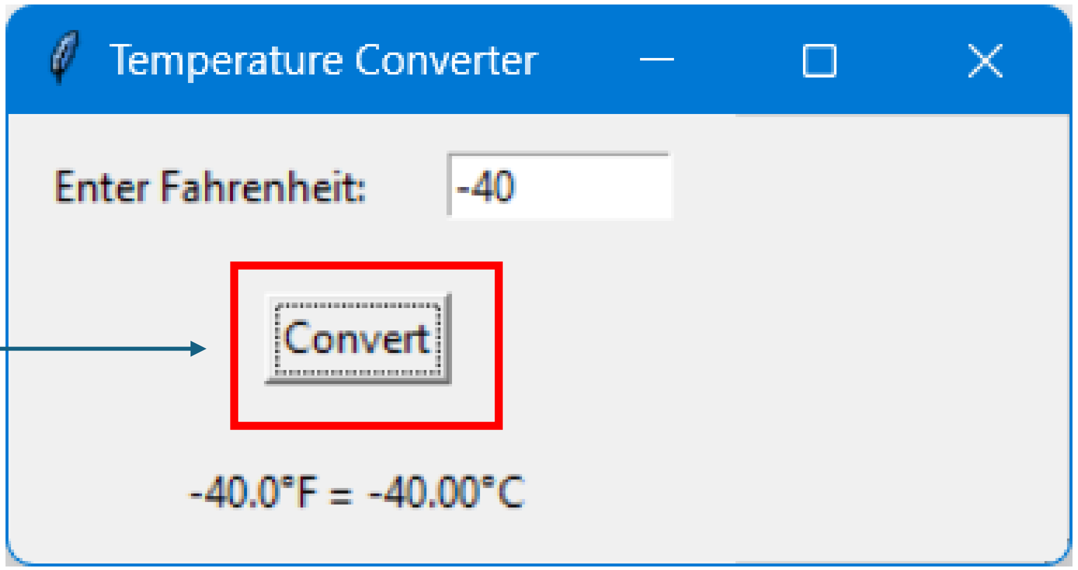
Explanation

- In the context of Python GUI programming, an event refers to any action or occurrence that happens within the graphical user interface, typically triggered by user interaction or system events.

- These events could include clicking a button, typing in a text field, moving the mouse, resizing a window, or any other action that causes a change in the state or behavior of the GUI application.

Events

A button (convert_button) to trigger the conversion function (convert), and a label (result_label) to display the conversion result.



-40.0°F = -40.00°C

Events

How events work in Tkinter

- **Event Binding**: Bind an event (such as button click) to a function (event handler). This is typically done using the command parameter for buttons or through methods like .bind() for other widgets or custom events.

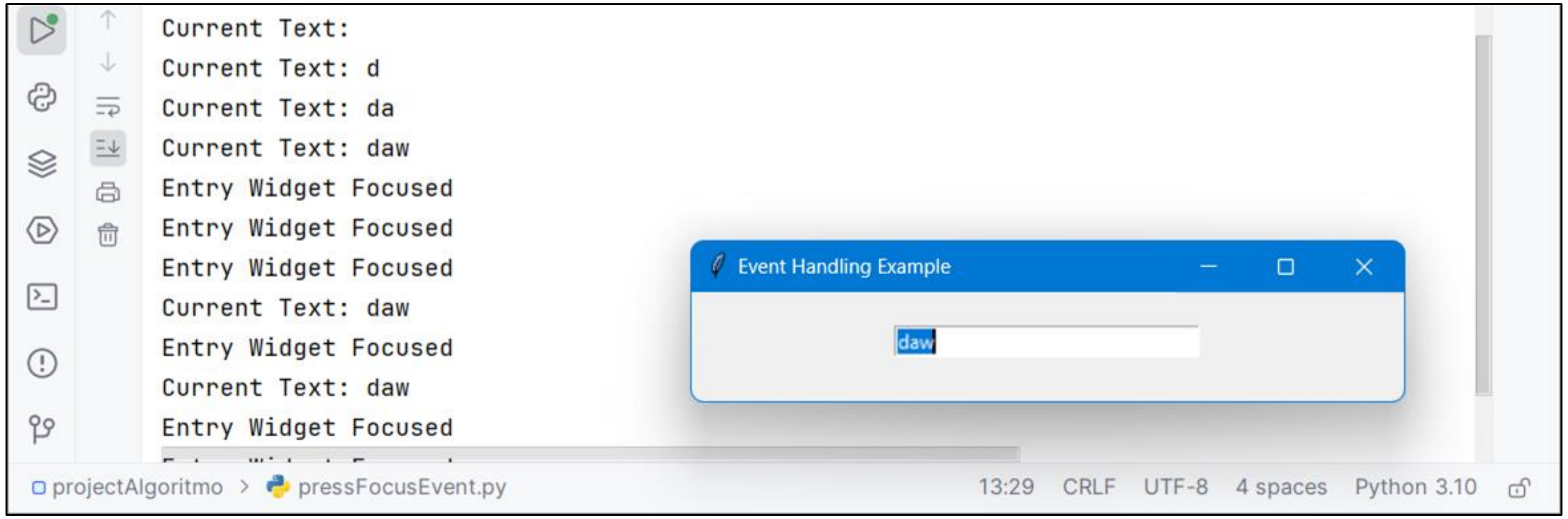
- **Event Handlers**: Event handlers (like on_button_click() in the example) are functions that execute in response to specific events. These functions can manipulate widgets, update data, or perform any action based on user interaction or system events.

- **Event Queue**: When an event (e.g., button click) occurs, Tkinter places the event in the queue. The event loop (mainloop()) retrieves events from the queue, dispatches them to the appropriate event handlers, and executes the associated functions.

Events

Example - Handling “key” and “focus” events

We will create a Tkinter window with an Entry widget. We'll handle events such as typing in the entry field (<Key> event) and gaining focus (<FocusIn> event).

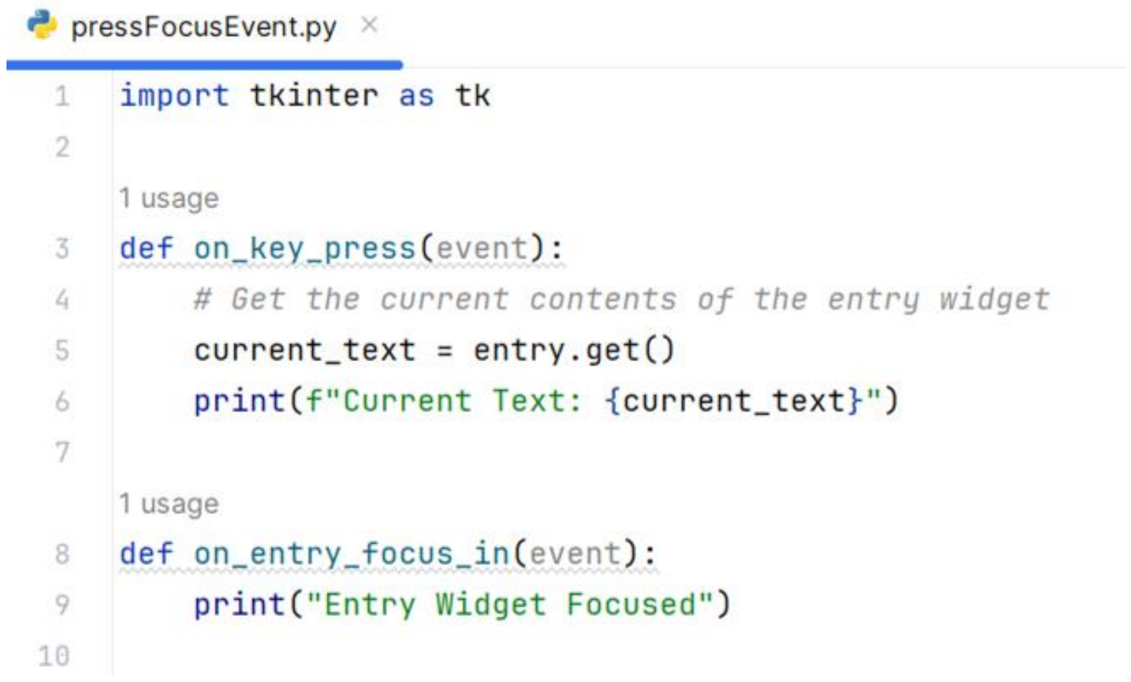


```
Current Text:
Current Text: d
Current Text: da
Current Text: daw
Entry Widget Focused
Entry Widget Focused
Entry Widget Focused
Current Text: daw
Entry Widget Focused
Current Text: daw
Entry Widget Focused
```

Event Handling Example
daw
projectAlgoritmo > pressFocusEvent.py
13:29 CRLF UTF-8 4 spaces Python 3.10

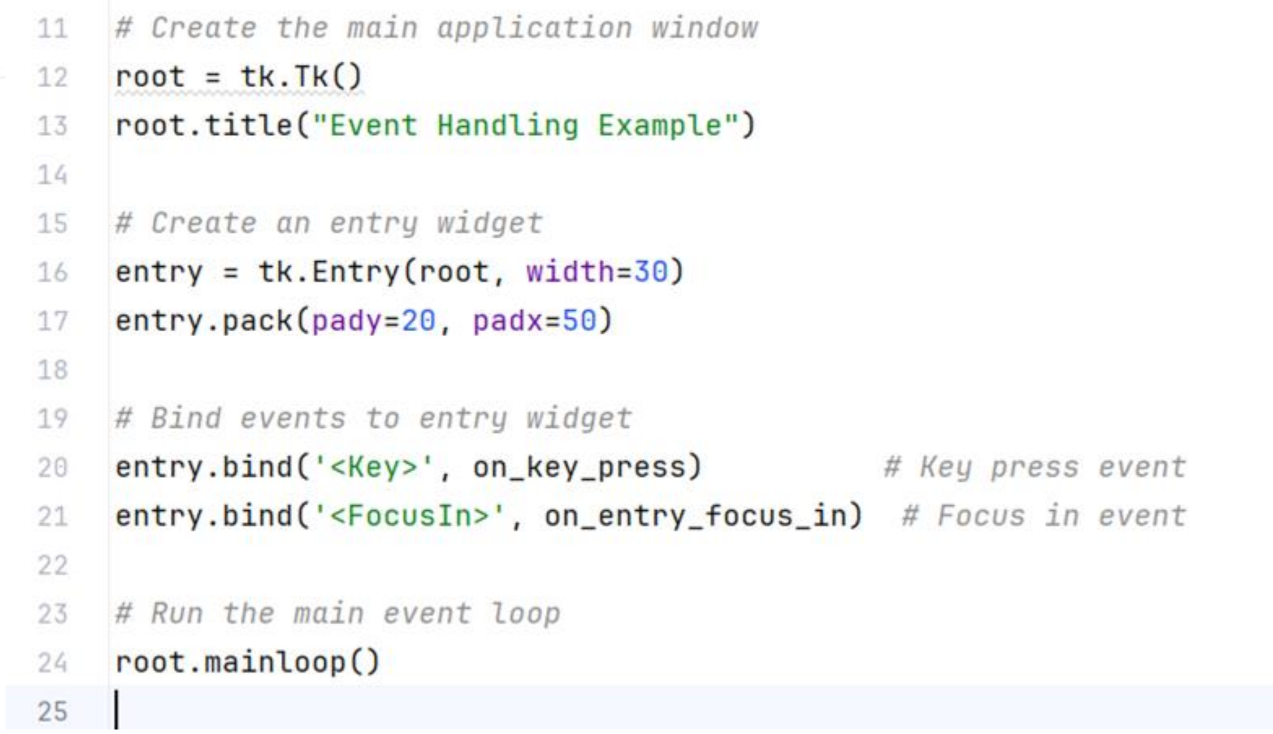
Events

Example – Code for handling “key” and “focus” events



pressFocusEvent.py

```python
import tkinter as tk

1 usage
def on_key_press(event):
    # Get the current contents of the entry widget
    current_text = entry.get()
    print(f"Current Text: {current_text}")

1 usage
def on_entry_focus_in(event):
    print("Entry Widget Focused")

```



```python
# Create the main application window
root = tk.Tk()
root.title("Event Handling Example")

# Create an entry widget
entry = tk.Entry(root, width=30)
entry.pack(pady=20, padx=50)

# Bind events to entry widget
entry.bind('<Key>', on_key_press)           # Key press event
entry.bind('<FocusIn>', on_entry_focus_in)  # Focus in event

# Run the main event loop
root.mainloop()

```

Events

Example – Explanation of handling “key” and “focus” events

Event Handling Functions

- on_key_press(event): This function handles the <Key> event, which occurs whenever a key is pressed while the entry widget has focus. It prints the current text in the entry widget to the console.

- on_entry_focus_in(event): This function handles the <FocusIn> event, which occurs when the entry widget gains focus. It simply prints a message to indicate that the widget has been focused.

Events

TRY IT

Create a GUI with Keyboard Shortcuts

Create a Tkinter GUI application that includes a text entry widget and uses keyboard shortcuts to interact with the text.

Implement the following features:

1. Pressing the Enter key while typing in the text entry widget should print the current text in the entry widget to the console.
2. Pressing the Escape key should clear the text in the entry widget.

Requirements:

1. Create a main window with a title "Keyboard Shortcuts Example".
2. Add an Entry widget to the window where users can type text.
3. Implement an event handler for the Enter key that prints the current text in the entry widget to the console.
4. Implement an event handler for the Escape key that clears the text in the entry widget.

7. Functions and Layouts

Functions and Layouts

Functions

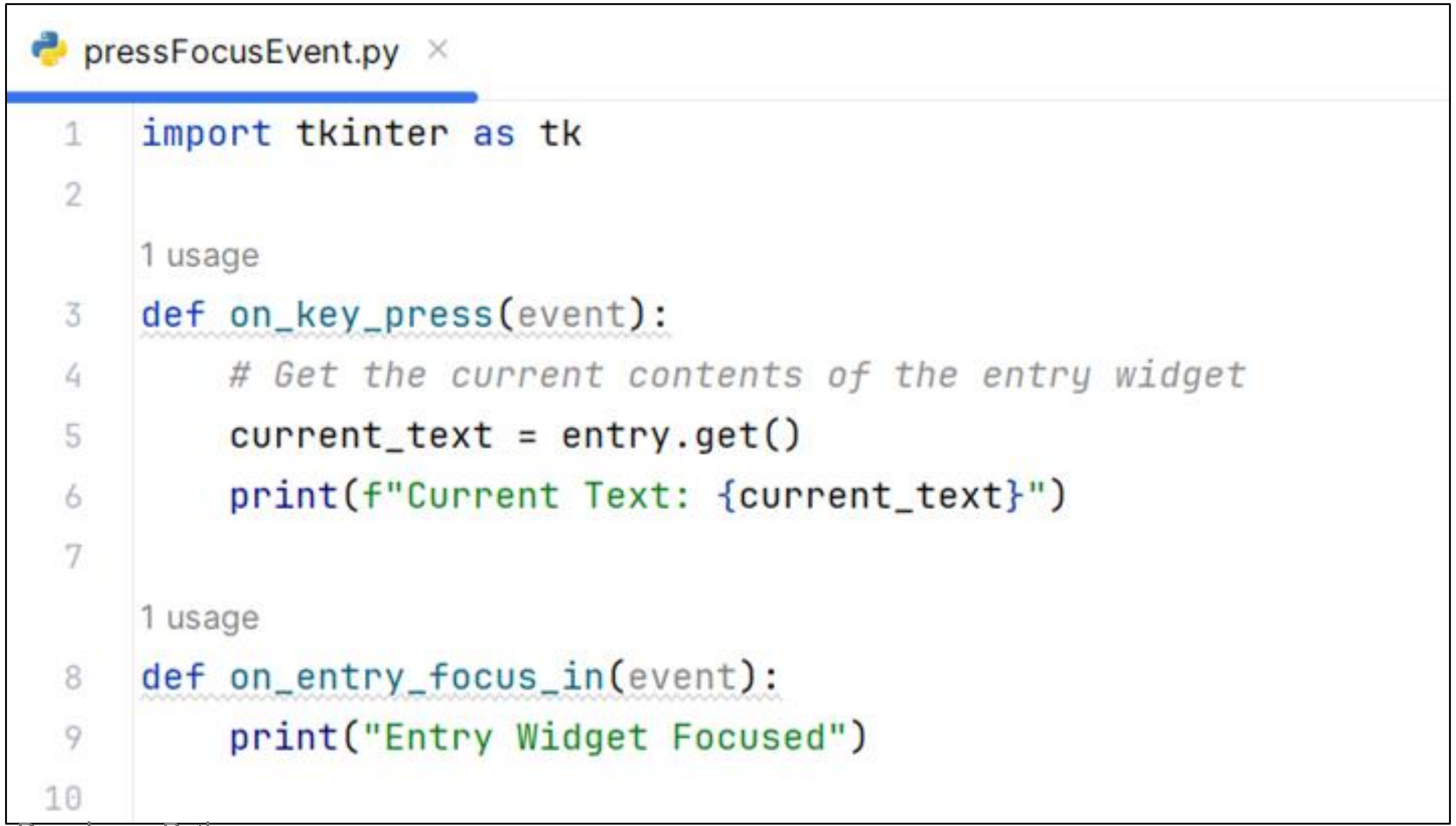
Functions in Tkinter are used to define the behavior and logic of your application. These functions are typically linked to events or widgets, such as button clicks, menu selections, or other user actions.

Need of function to work with Tkinter?

- Event Handling : Functions allow you to define how your application should respond to specific events, such as button clicks, key presses, or mouse movements.
- Modularity : Functions make your code modular by breaking it into smaller, manageable pieces. This modularity allows you to test and debug individual parts of your application separately.
- State Management : Functions can help manage the state of your application. For example, you can create functions that update the content of widgets or variables based on user interactions.
- Reusability : Functions promote reusability. Once you define a function, you can reuse it across different parts of your application or even in different applications.

Functions and Layouts

We have already used on_keypress() and on_entry_focus_in() functions to handle key press and get fucus events.



pressFocusEvent.py

```python
import tkinter as tk

1 usage
def on_key_press(event):
    # Get the current contents of the entry widget
    current_text = entry.get()
    print(f"Current Text: {current_text}")

1 usage
def on_entry_focus_in(event):
    print("Entry Widget Focused")

```

Functions and Layouts

Layouts

Explanation

In Tkinter, a layout refers to the way widgets (such as buttons, labels, text boxes, etc.) are organized within a window or frame.

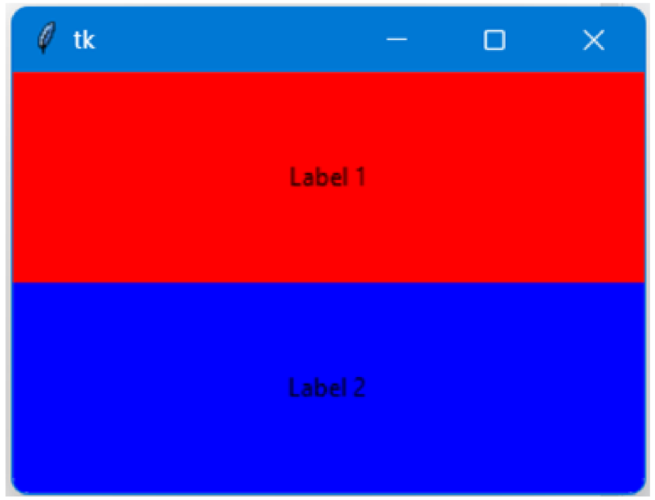
Type of layouts

Tkinter offers several ways to place widgets in the window, the most commons are -

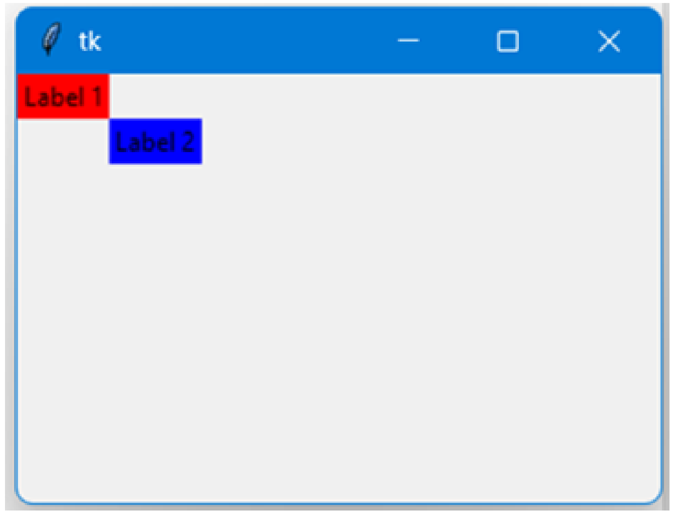
- Pack layout
- Grid layout
- Place layout

Functions and Layouts

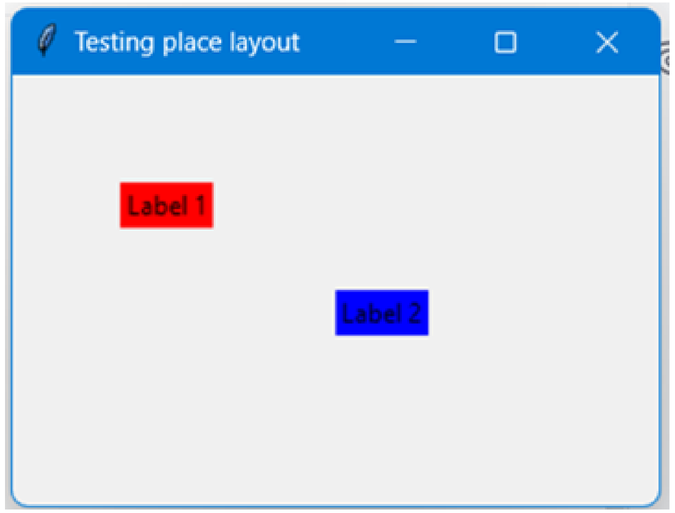
Layouts



Pack layout



Grid layout



Place layout

Functions and Layouts

TRY IT

Create a GUI with different layouts and functions.

Create a Tkinter GUI of Registration form. Collect data from user inputs – name, email, mobile number, age, gender, hobbies, and address, and display these into terminal.

Instructions:

1.Use functions to handle various events corresponding to widgets.

2.Use appropriate layout to organize the widgets.

3.On click of the “Submit” button all the data inputted by user should be display on terminal.

4. Use a variety of widgets.

Question?

Thank you