Python GUI – using Tkinter

Python is a versatile language and it can be used to create Graphical User Interfaces as well. This tutorial covers essentials of GUI programming in Python using Tkinter framework.

Tkinter is built into the Standard Python Library. Its cross – platform and uses native operating system elements. It can work with Windows, macOS and Linux as well.

The only drawback it its old styled and GUI looks quit out dated.

In this tutorial we will learn:

- **How to setup Tkinter GUI App**
- **Setting layout Part – 1: Window**
- **How to add elements**
- **How to change attributes of different elements (like styling etc.)?**
- **How to handle events (Adding actions) and take decisions based on user data.**
- **Sample App – Temperature Conversion**
- **GUI Application with Data Storage.**

1. **<u>How to setup Tkinter GUI App</u>**
    a. First of all, we have to make sure Tkinter is installed.
    b. Start a new Python Project using VS Code and IDE of your choice.
    c. Import Tkinter

```
import tkinter as tk
```

    d. If you don't see an error, that means Tkinter is installed. Here **tk** is allies. (a short name to remember).
    e. Also, you can use this command to get rid of writing tk all the time with code.

```
from tkinter import *
```

    f. NOTE: Its totally up to you to which way to follow. But I prefer the first method. Anyways.
    g. Now we need to create a window first. In order to create a window, we need the Tk() class.

```
window = tk.Tk()
```

    h. In the line above, we have created an instance of the Tk() class with the variable name window.
    i. Once the window is created, we have to keep it open, we use the mainloop() method.

```
window.mainloop()
```

    j. window.mainloop() tells Python to run the Tkinter event loop. This method listens for events, such as button clicks or keypresses, and blocks any code that comes after it from running until you close the window where you called the method.

k. So now the program should look like this:

```
import tkinter as tk
window = tk.Tk()
window.mainloop()
```

l. Now run the code you will see a window Popping up.

m. This is how you set up a GUI. Only three lines!

2. **Setting layout Part – 1: Window**

a. Now we will first add a title to the Main Window

```
window.title("GUI Tutorial")
```

b. Note that the size of the window is flexible. We can also set Width and height of the window by using geometry method of the Tk class.

c. This code will create a window of Width 800 and height 400.

```
window.geometry("800x400")
```

d. If we want to display our window at a certain position of our desktop, we can add co-ordinates of the starting points as well. For example:

```
window.geometry("800x400+500+200")
```

e. Here 500 sets the x – coordinate and 200 sets the y – coordinate.

f. You can also decide if a window is resizable or not. For example:

```
window.resizable(True, False)
```

g. Above statement will allow horizontal resizing but will not resize vertically. You can similarly do adjustments. (By default both are true).

h. Also, you can specify the minimum and maximum size constraints for the window using the minsize() and maxsize() methods, respectively.

i. Want to set a custom icon(favicon)? No problem!

```
window.iconphoto(True, tk.PhotoImage(file="icon.png"))
```

j. Iconphoto is a method of the Tk class in Tkinter used to set the icon (or favicon) of the window.

k. **NOTE:** When you pass True as the first argument, it indicates that the image passed as the second argument should be used as both the window icon (the small icon displayed in the title bar of the window) and the application icon (the icon displayed in the taskbar or system tray when the window is minimized).

l. You can set the initial state of the window (e.g., minimized, maximized) using the state() method.

```
window.state("zoomed")
```

m. If you want the application to run as minimized window you can use “iconic” instead of “zoomed”.

n. Finally, you can also add some styling using configure() method.

o. Change background color of the window like this:

```
window.configure(bg="black")
```

p. Also, we change the border widths as well.

```
window.configure(highlightbackground="blue", highlightthickness=1)
```

q. NOTE: The highlightbackground is being set to "blue", which means that the border color surrounding the window will be blue. This border is typically visible

when the window does not have the focus or when it's not active. It's often used for aesthetic purposes or to provide visual feedback to the user about the focus state of the window.

r. This option sets the width of the focus highlight border. A value of 0 means no highlight border.

3. **How to add elements**
    a. In this part, we will learn how to display some common elements.
        i. Label
        ii. Entry Field
        iii. Check Box
        iv. Radio Buttons
        v. List Box
        vi. Buttons
    b. Labels:
        i. Labels are text that are displayed on Windows or frames.

```
label = tk.Label(window, text="Hello, World!")
label.pack()
```

        ii. Here, window is where the label is to be displayed.
        iii. The **pack()** method in Tkinter is used to automatically size and position widgets within a container.
    c. Entry:
        i. Entry fields allow users to input text. You can add an entry field using the Entry widget.

```
entry = tk.Entry(window)
entry.pack()
```

        ii. NOTE: In Tkinter, the Entry widget does not natively support a placeholder text (a default text that disappears when the user starts typing).
    d. Check Box:
        i. Checkbuttons allow users to toggle between selected and deselected states. You can add a checkbutton using the Checkbutton widget.

```
checkbutton = tk.Checkbutton(window, text="Check me!")
checkbutton.pack()
```

        ii. This will place a checkbox on the window.
    e. Radio Buttons:
        i. Radio buttons allow users to select one option from a group of options. You can add radio buttons using the Radiobutton widget.

```
ops = ""
radio_button1 = tk.Radiobutton(window, text="Option 1", value=1,
variable=ops)
radio_button1.pack()
radio_button2 = tk.Radiobutton(window, text="Option 2", value=2,
variable=ops)
```

```
radio_button2.pack()
```

ii. This will place two Radio Buttons with text shown as well. Note that extra parameters **value** and **variables**.

1. Value is the value associated with a particular radio button.
2. Variable is the value attached to a particular variable. Since Radio buttons are used to choose one option from the given options.

f. List Box:

i. Listboxes allow users to select one or more items from a list. You can add a listbox using the Listbox widget.

```
listbox = tk.Listbox(window)
listbox.pack()
```

ii. This will create an empty list box. However you can add options to list boxes like this:

```
listbox.insert(tk.END, "Option 1")
listbox.insert(tk.END, "Option 2")
listbox.insert(tk.END, "Option 3")
```

iii. Now our list box is having three options.

g. Buttons:

i. A button is a widget that can contain text and can perform an action when clicked. Here is how to add it

```
button = tk.Button(text="Click me")
button.pack()
```

ii. Buttons are usually associated with actions (functions that are called when buttons are pressed). This is done by the **command** attribute.

```
def my_function():
    pass

button = tk.Button(text="Click me", command=my_function)
button.pack()
```

iii. Note the function needs to be defined first in order to use it inside the buttons.

4. <u>How to change attributes of different elements?</u>

a. In Tkinter, you can change the appearance and behavior of different elements (widgets) by configuring their various attributes. Here are some common attributes you can modify:

i. Background and foreground colors
ii. Font (type and size)
iii. Width and Height
iv. Border Color and width
v. Relief
vi. Text Alignment

b. **Widget.configu()**:

i. The config() method is used to modify properties of widgets.

ii. **Examples:**

1. **widget.config(bg="red")** # bg changes the background color.
2. **widget.config(fg="blue")** # fg changes the foreground color.
3. **widget.config(font=("Arial", 12))** # changes the fonts
4. **widget.config(width=20, height=2)** # changes width and height
5. **widget.config(highlightbackground="green")** # Sets the color of the border surrounding the widget.
6. **widget.config(highlightthickness=2)** # Sets the width of the border surrounding the widget.
7. **widget.config(relief="sunken")** # Specifies the type of border to draw around the widget. Options include "flat", "sunken", "raised", "ridge", and "groove".
8. **widget.config(justify="center")** # Sets the text alignment within the widget (for labels and buttons).

iii. **Practice:**

```
another = tk.Entry()
another.config(bg="blue", fg="white", width=20,
               font=("Verdana, 12"), relief="flat", justify="center")
another.pack()
```

c. NOTE: These are just some general characteristics, more properties can be accessed and changed but for that Tkinter documentation can be a good resource.

5. **<u>Setting layout Part – 2: Geometry Managers</u>**

a. Geometry managers are used to control the layouts (placement of widgets on the window).

b. Following methods are used:

i. .pack()
ii. .place()
iii. .grid()

c. Let's discuss them.

d. Pack() Geometry Manager:

i. The .pack() geometry manager uses a packing algorithm to place widgets in a Frame or window in a specified order. It's a simple way to position widgets within a container (such as a window or frame) using either horizontal or vertical packing.

ii. Default packing is vertical. So if we pack two widgets in a window or a container, they will be stacked vertically.

iii. In the examples above we can see that all the items are stacked vertically.

iv. But if you want to stack some widgets horizontally, let's try this:

```
frame = tk.Frame()
frame.pack()
```

```
label1 = tk.Label(frame, text="Enter your country: ")
label1.pack(side="left")

entry1 = tk.Entry(frame)
entry1.pack(side="left")
```

v. In this example, two widgets have been stacked horizontally.
vi. **Pack()** accepts some keywords – arguments more precise placement of widgets.
vii. you can set the fill keyword argument to specify in which direction the frames should fill. The options are **tk.X** to fill in the horizontal direction, **tk.Y** to fill vertically, and **tk.BOTH** to fill in both directions.

```
frame2 = tk.Frame(master=window, height=50, bg="yellow")
frame2.pack(fill=tk.X)
```

viii. Notice that the width is not set on the Frame widgets. Width is no longer necessary because each frame sets **.pack()** to fill horizontally, overriding any width you may set.
ix. One of the nice things about filling the window with .pack() is that the fill is responsive to window resizing.
x. The **side** keyword argument of .pack() specifies on which side of the window the widget should be placed. These are the available options:
    1. **tk.TOP** (or top, bottom, left, right)
    2. **tk.BOTTOM**
    3. **tk.LEFT**
    4. **tk.RIGHT**
xi. If you don't set side, then .pack() will automatically use tk.TOP and place new widgets at the top of the window, or at the topmost portion of the window that isn't already occupied by a widget.
xii. Note: If you stack widgets horizontally, fill them in Y axis will make the widgets responsive.
xiii. Expend keyword make the widgets expandable if set to true.

e. Place() Geometry Manager:

i. You can use .place() to control the precise location that a widget should occupy in a window or Frame. You must provide two keyword arguments, x and y, which specify the x- and y-coordinates for the top-left corner of the widget. Both x and y are measured in pixels, not text units.
ii. Keep in mind that the origin, where x and y are both 0, is the top-left corner of the Frame or window. So, you can think of the y argument of .place() as the number of pixels from the top of the window, and the x argument as the number of pixels from the left edge of the window.
iii. <u>NOTE:</u>
    1. Layout can be difficult to manage with .place(). This is especially true if your application has lots of widgets.

2. Layouts created with .place() aren't responsive. They don't change as the window is resized.

iv. Example:

```
import tkinter as tk

window = tk.Tk()

frame = tk.Frame(master=window, width=150, height=150)
frame.pack()

label1 = tk.Label(master=frame, text="I'm at (0, 0)", bg="red")
label1.place(x=0, y=0)

label2 = tk.Label(master=frame, text="I'm at (75, 75)", bg="yellow")
label2.place(x=75, y=75)

window.mainloop()
```

v. NOTE: One of the main challenges of cross-platform GUI development is making layouts that look good no matter which platform they're viewed on, and .place() is a poor choice for making responsive and cross-platform layouts.

f. <u>Grid() Geometry Manager:</u>

i. The grid() method allows you to create a grid-like layout where widgets are placed in rows and columns.

ii. .grid() works by splitting a window or Frame into rows and columns. You specify the location of a widget by calling .grid() and passing the row and column indices to the row and column keyword arguments, respectively.

iii. Both row and column indices start at 0, so a row index of 1 and a column index of 2 tells .grid() to place a widget in the third column of the second row.

iv. This is how you generate a grid:

```
import tkinter as tk

window = tk.Tk()
window.title("2x2 Grid Example")

# Create widgets
label1 = tk.Label(window, text="Row 0, Column 0")
label2 = tk.Label(window, text="Row 0, Column 1")
label3 = tk.Label(window, text="Row 1, Column 0")
label4 = tk.Label(window, text="Row 1, Column 1")

# Place widgets in the grid
label1.grid(row=0, column=0)
```

```
label2.grid(row=0, column=1)
label3.grid(row=1, column=0)
label4.grid(row=1, column=1)

window.mainloop()
```

v. We can also use loops to generate grids. Such as:

```
import tkinter as tk

window = tk.Tk()
window.title("3x3 Grid Example")

# Create a 3x3 grid of frames with borders
for i in range(3):  # Rows
    for j in range(3):  # Columns
        frame = tk.Frame(window, borderwidth=2,
                         relief="ridge", width=100, height=100)
        frame.grid(row=i, column=j)
        label = tk.Label(master=frame, text=f"Row {i}\nColumn {j}")
        label.pack()

window.mainloop()
```

vi. In the above example, we are using two geometry managers:
    1. Each frame is attached to window with the .grid() geometry manager.
    2. Each label is attached to its master Frame with .pack()

vii. We can also control the internal and external spacing around widgets using padding.
    1. For external padding, we use **padx** and **pady**.
    2. External padding adds some space around the outside of a grid cell.
    3. Padding is measured in pixels.
    4. Replace:

```
frame.grid(row=i, column=j)
```

    5. With:

```
frame.grid(row=i, column=j, padx=5, pady=5)
```

    6. Note: We can also use padding with .pack(). Try updating the following line:

```
label.pack(padx=5, pady=5)
```

        a. The extra padding around the Label widgets gives each cell in the grid a little bit of breathing room between the Frame border and the text in the label.
    7. If we increase size of the window, the grid does not expand. We can set the expansion by columnconfigure() and rowcofigure() on

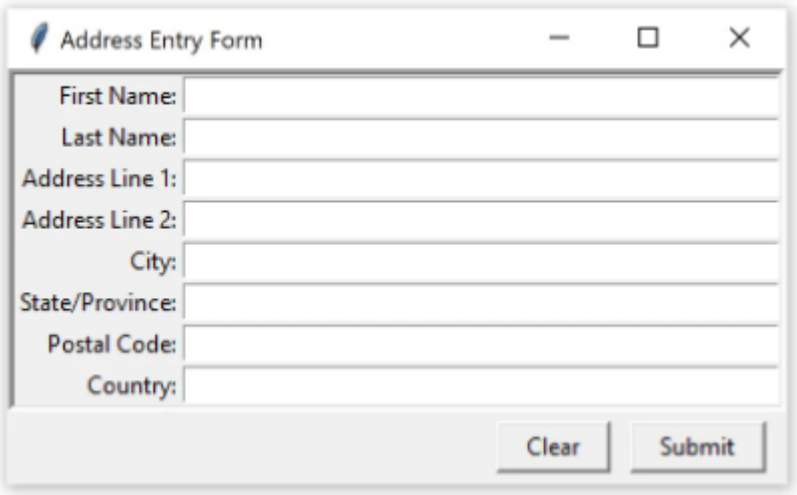
the window object. (On window, because the grid is attached to the window, always).

8. Both .columnconfigure() and .rowconfigure() take three essential arguments:
    a. Index: The index of the grid column or row that you want to configure or a list of indices to configure multiple rows or columns at the same time
    b. Weight: It's like the growth factor. A keyword argument called weight that determines how the column or row should respond to window resizing, relative to the other columns and rows.
        i. Weight is set to 0 by default, which means that the column or row doesn't expand as the window resizes.
        ii. If every column or row is given a weight of 1, then they all grow at the same rate.
        iii. If one column has a weight of 1 and another a weight of 2, then the second column expands at twice the rate of the first.
    c. Minimum Size: A keyword argument called minsize that sets the minimum size of the row height or column width in pixels.
    d. Adjust the code, add these two new lines in the outer loop:

```
window.columnconfigure(i, weight=1, minsize=75)
window.rowconfigure(i, weight=1, minsize=50)
```

9. By default, widgets are centered in their grid cells.
    a. You can change the location of each label inside of the grid cell using the sticky parameter, which accepts a string containing one or more of the following letters:
    b. "n" or "N" to align to the top-center part of the cell
    c. "e" or "E" to align to the right-center side of the cell
    d. "s" or "S" to align to the bottom-center part of the cell
    e. "w" or "W" to align to the left-center side of the cell
    f. The letters "n", "s", "e", and "w" come from the cardinal directions north, south, east, and west.
    g. NOTE THAT: Just like we have fill property in .pack(), similarly we have sticky property in .grid().

viii. Example: Let's build this:



ix.

x. Solution can be provided.

6. **How to handle events (Adding actions) and take decisions based on user data.**
   a. In this section, you'll learn how to bring your applications to life by performing actions whenever certain events occur.
   b. NOTE: When you create a Tkinter application, you must call window.mainloop() to start the event loop. During the event loop, your application checks if an event has occurred. If so, then it'll execute some code in response.
   c. In Tkinter, you write functions called event handlers for the events that you use in your application.
   d. An event is any action that occurs during the event loop that might trigger some behavior in the application, such as when a key or mouse button is pressed.
   e. Bind() Method:
      i. The bind() method in Tkinter is used to attach event handlers to widgets. This method allows you to specify a callback function that gets executed when a particular event occurs on the widget.
      ii. The procedure to use bind() goes like this:
         1. **Define a Function:** First, you define a Python function that you want to be called when a specific event occurs on the widget.
         2. **Create a Widget:** Next, you create a Tkinter widget (e.g., Button, Entry, Canvas, etc.) to which you want to attach the event handler.
         3. **Bind the Function to the Widget's Event:** Then, you use the bind() method on the widget to attach the defined function to a particular event of the widget.
         4. **Example:**

```
import tkinter as tk

def handle_click(event):
    print("Button clicked!")
```

```
window = tk.Tk()
window.title("Bind Example")

button = tk.Button(window, text="Click me!")
button.pack()

# Bind the handle_click function to the button's <Button-1> event
(left mouse button click)
button.bind("<Button-1>", handle_click)

window.mainloop()
```

5. We define a function handle_click that prints "Button clicked!" to the console.
6. We create a Tkinter button widget named button and pack it into the main window.
7. We use the bind() method to attach the handle_click function to the <Button-1> event of the button widget. The <Button-1> event represents a left mouse button click. When this event occurs on the button, the handle_click function will be called.
8. There are other events for mouse button clicks, including "<Button-2>" for the middle mouse button and "<Button-3>" for the right mouse button.

iii. Using Command:

1. Every Button widget has a command attribute that you can assign to a function. Whenever the button is pressed, the function is executed.
2. In the following example, we are using the increase and decrease functions to update the values of the displayed text.

```
import tkinter as tk


def increase():
    value = int(lbl_value["text"])
    lbl_value["text"] = f"{value + 1}"


def decrease():
    value = int(lbl_value["text"])
    lbl_value["text"] = f"{value - 1}"


window = tk.Tk()

window.rowconfigure(0, minsize=50, weight=1)
window.columnconfigure([0, 1, 2], minsize=50, weight=1)
```

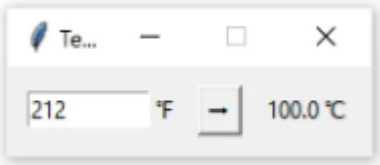
```
btn_decrease = tk.Button(master=window, text="-", command=decrease)
btn_decrease.grid(row=0, column=0, sticky="nsew")

lbl_value = tk.Label(master=window, text="0")
lbl_value.grid(row=0, column=1)

btn_increase = tk.Button(master=window, text="+", command=increase)
btn_increase.grid(row=0, column=2, sticky="nsew")


window.mainloop()
```

3. In the above examples, the command attribute call backs the increase and decrease functions.

f. Practice: Can you simulate a rolling dice?

7. Sample App – Temperature Conversion

a. Source Code is provided.

b. 

8. References:

a. https://tkdocs.com/tutorial/grid.html
b. https://tkdocs.com/tutorial/index.html
c. https://realpython.com/python-gui-tkinter/