

Practical Learning #8

Python GUI - tkinter

Python has a lot of GUI frameworks, but Tkinter is the only framework that's built into the Python standard library. Tkinter has several strengths. It's cross-platform, so the same code works on Windows, macOS, and Linux. Visual elements are rendered using native operating system elements, so applications built with Tkinter look like they belong on the platform where they're run.

Although Tkinter is considered the de-facto Python GUI framework, it's not without criticism. One notable criticism is that GUIs built with Tkinter look outdated. If you want a shiny, modern interface, then Tkinter may not be what you're looking for.

However, Tkinter is lightweight and relatively painless to use compared to other frameworks. This makes it a compelling choice for building GUI applications in Python, especially for applications where a modern sheen is unnecessary, and the top priority is to build something that's functional and cross-platform quickly.

Building Your First Python GUI Application With Tkinter

The foundational element of a Tkinter GUI is the **window**. Windows are the containers in which all other GUI elements live. These other GUI elements, such as text boxes, labels, and buttons, are known as **widgets**. Widgets are contained inside of windows.

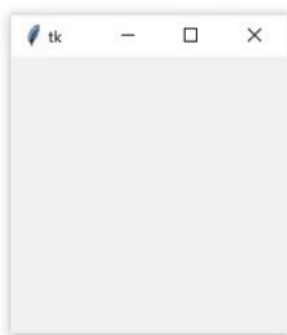
1. Create a new python file named PL8_LastnameFirstname.
2. Import the python GUI Tkinter module. Type the following:

```
import tkinter
```

3. A **window** is an instance of Tkinter's `Tk` class. Let's create a new window and assign it to the variable `window`.

```
window = tkinter.Tk()
```

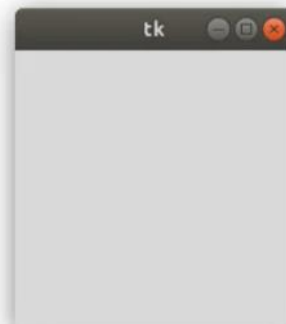
4. When you execute the above code, a new window pops up on your screen. How it looks depends on your operating system:



(a) Windows



(b) macOS



(c) Ubuntu

Adding a Widget

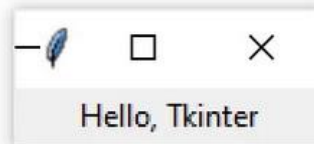
5. Now that you have a window, you can add a widget. Use the `tkinter.Label` class to add some text to a window. Create a Label widget with the text "Hello, World" and assign it to a variable called `greeting`:

```
greeting = tkinter.Label(text="Hello, World")
```

6. The window you created earlier doesn't change. You just created a Label widget, but you haven't added it to the window yet. There are several ways to add widgets to a window. Right now, you can use the Label widget's `.pack()` method:

```
greeting.pack()
```

The window now looks like this:



When you `.pack()` a widget into a window, Tkinter sizes the window as small as it can while still fully encompassing the widget.

7. Add the following code and execute:

```
window.mainloop()
```

Nothing seems to happen, but notice that a new prompt does not appear in the shell.

`window.mainloop()` tells Python to run the Tkinter **event loop**. This `mainloop()` method listens for events, such as button clicks or keypresses, and **blocks** any code that comes after it from running until the window it's called on is closed. `mainloop()` is an infinite loop used to run the application, wait for an event to occur and process the event as long as the window is not closed. Close the window you've created, and you'll see a new prompt displayed in the shell.

Working with Containers

Canvas: It is used to draw pictures and other complex layout like graphics, text and widgets. The general syntax is:

```
w = Canvas(master, option=value)
```

`master` is the parameter used to represent the parent window.

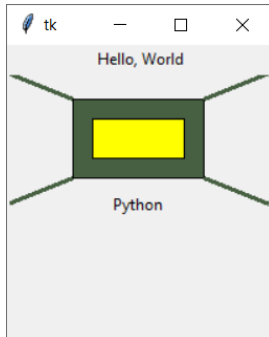
There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

- **bd:** to set the border width in pixels.
- **bg:** to set the normal background color.
- **cursor:** to set the cursor used in the canvas.
- **highlightcolor:** to set the color shown in the focus highlight.
- **width:** to set the width of the widget.
- **height:** to set the height of the widget.

8. Add the following code before `window.mainloop()`

```
w = tkinter.Canvas(window, width=200, height=200)
w.pack()
canvas_height=200
canvas_width = 200
y = int(canvas_height/2)
w.create_rectangle(50, 20, 150, 80, fill="#476042")
w.create_rectangle(65, 35, 135, 65, fill="yellow")
w.create_line(0, 0, 50, 20, fill="#476042", width=3)
w.create_line(0, 100, 50, 80, fill="#476042", width=3)
w.create_line(150,20, 200, 0, fill="#476042", width=3)
w.create_line(150, 80, 200, 100, fill="#476042", width=3)
w.create_text(y,y,text="Python")
```

Output:



Frame: It acts as a container to hold the widgets. It is used for grouping and organizing the widgets. The general syntax is:

```
w = Frame(master, option=value)
master is the parameter used to represent the parent window.
```

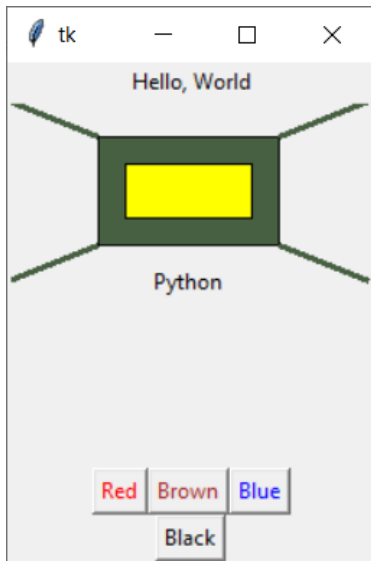
There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

- **highlightcolor:** To set the color of the focus highlight when widget has to be focused.
- **bd:** to set the border width in pixels.
- **bg:** to set the normal background color.
- **cursor:** to set the cursor used.
- **width:** to set the width of the widget.
- **height:** to set the height of the widget.

9. Add the following codes before `window.mainloop()`:

```
frame = tkinter.Frame(window)
frame.pack()
bottomframe = tkinter.Frame(window)
bottomframe.pack( side = tkinter.BOTTOM )
redbutton = tkinter.Button(frame, text = 'Red', fg = 'red')
redbutton.pack( side = tkinter.LEFT)
greenbutton = tkinter.Button(frame, text = 'Brown', fg='brown')
greenbutton.pack( side = tkinter.LEFT )
bluebutton = tkinter.Button(frame, text = 'Blue', fg = 'blue')
bluebutton.pack( side = tkinter.LEFT )
blackbutton = tkinter.Button(bottomframe, text = 'Black', fg = 'black')
blackbutton.pack( side = tkinter.BOTTOM)
```

Output:



Working with Layouts

tkinter also offers access to the geometric configuration of the widgets which can organize the widgets in the parent windows. There are mainly three geometry manager classes class.

1. **pack() method:** It organizes the widgets in blocks before placing in the parent widget.
2. **grid() method:** It organizes the widgets in grid (table-like structure) before placing in the parent widget.
3. **place() method:** It organizes the widgets by placing them on specific positions directed by the programmer.

The three layout managers pack, grid, and place should never be mixed in the same master window! Geometry managers serve various functions. They:

- arrange widgets on the screen
- register widgets with the underlying windowing system
- manage the display of widgets on the screen

Arranging widgets on the screen includes determining the size and position of components. Widgets can provide size and alignment information to geometry managers, but the geometry managers has always the final say on the positioning and sizing.

Pack

Pack is the easiest to use of the three geometry managers of Tk and Tkinter. Instead of having to declare precisely where a widget should appear on the display screen, we can declare the positions of widgets with the pack command relative to each other. The pack command takes care of the details. Though the pack command is easier to use, this layout managers is limited in its possibilities compared to the grid and place mangers. For simple applications it is definitely the manager of choice. For example, simple applications like placing a number of widgets side by side, or on top of each other.

1. Create a new python file named PL8Pack_LastnameFirstname.
2. Type the following:

```
import tkinter as tk

root = tk.Tk()

w = tk.Label(root, text="Red Sun", bg="red", fg="white")
w.pack()
w = tk.Label(root, text="Green Grass", bg="green", fg="black")
w.pack()
w = tk.Label(root, text="Blue Sky", bg="blue", fg="white")
w.pack()

tk.mainloop()
```

Output:



fill Option

In our example, we have packed three labels into the parent widget "root". We used pack() without any options. So pack had to decide which way to arrange the labels. As you can see, it has chosen to place the label widgets on top of each other and centre them. Furthermore, we can see that each label has been given the size of the text. If you want to make the widgets as wide as the parent widget, you have to use the fill=X option:

3. Edit your code as shown below:

```
import tkinter as tk

root = tk.Tk()

w = tk.Label(root, text="Red Sun", bg="red", fg="white")
w.pack(fill=tk.X)
w = tk.Label(root, text="Green Grass", bg="green", fg="black")
w.pack(fill=tk.X)
w = tk.Label(root, text="Blue Sky", bg="blue", fg="white")
w.pack(fill=tk.X)



tk.mainloop()
```



Output:



Padding

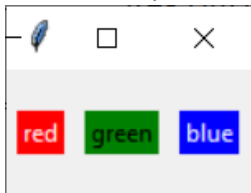
The pack() manager knows four padding options, i.e. internal and external padding and padding in x and y direction:

padx	<p>External padding horizontally:</p>  <p>Code:</p> <pre>import tkinter as tk root = tk.Tk() w = tk.Label(root, text="Red Sun", bg="red", fg="white") w.pack(fill=tk.X, padx=10) w = tk.Label(root, text="Green Grass", bg="green", fg="black") w.pack(fill=tk.X, padx=10) w = tk.Label(root, text="Blue Sky", bg="blue", fg="white") w.pack(fill=tk.X, padx=10) tk.mainloop()</pre>
pady	<p>External padding, vertically</p>  <p>Code:</p> <pre>import tkinter as tk root = tk.Tk() w = tk.Label(root, text="Red Sun", bg="red", fg="white") w.pack(fill=tk.X, pady=10) w = tk.Label(root, text="Green Grass", bg="green", fg="black") w.pack(fill= tk.X, pady=10) w = tk.Label(root, text="Blue Sky", bg="blue", fg="white") w.pack(fill=tk.X, pady=10) tk.mainloop()</pre>

ipadx	<p>Internal padding, horizontally.</p> <p>In the following example, we change only the label with the text "Green Grass", so that the result can be easier recognized. We have also taken out the fill option.</p>  <p>Code:</p> <pre>import tkinter as tk root = tk.Tk() w = tk.Label(root, text="Red Sun", bg="red", fg="white") w.pack() w = tk.Label(root, text="Green Grass", bg="green", fg="black") w.pack(ipadx=10) w = tk.Label(root, text="Blue Sky", bg="blue", fg="white") w.pack() tk.mainloop()</pre>
ipady	<p>Internal padding, vertically</p> <p>We will change the last label of our previous example to ipady=10.</p>  <p>Code:</p> <pre>import tkinter as tk root = tk.Tk() w = tk.Label(root, text="Red Sun", bg="red", fg="white") w.pack() w = tk.Label(root, text="Green Grass", bg="green", fg="black") w.pack(ipadx=10) w = tk.Label(root, text="Blue Sky", bg="blue", fg="white") w.pack(ipady=10) tk.mainloop()</pre>
<p>The default value in all cases is 0.</p>	

Placing widgets side by side

We want to place the three labels side by side now and shorten the text slightly:



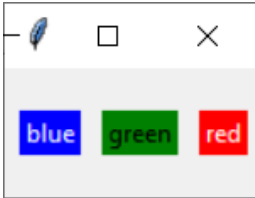
The corresponding code looks like this:

```
import tkinter as tk

root = tk.Tk()

w = tk.Label(root, text="red", bg="red", fg="white")
w.pack(padx=5, pady=10, side=tk.LEFT)
w = tk.Label(root, text="green", bg="green", fg="black")
w.pack(padx=5, pady=20, side=tk.LEFT)
w = tk.Label(root, text="blue", bg="blue", fg="white")
w.pack(padx=5, pady=20, side=tk.LEFT)
tk.mainloop()
```

If we change LEFT to RIGHT in the previous example, we get the colours in reverse order:



Place Geometry Manager

The Place geometry manager allows you explicitly set the position and size of a window, either in absolute terms, or relative to another window. The place manager can be accessed through the place method. It can be applied to all standard widgets.

We use the place geometry manager in the following example. We are playing around with colours in this example, i.e. we assign to every label a different colour, which we randomly create using the randrange method of the random module. We calculate the brightness (grey value) of each colour. If the brightness is less than 120, we set the foreground colour (fg) of the label to White otherwise to black, so that the text can be easier read.

1. Create a new python file named PL8Place_LastnameFirstname.
2. Type the following code:

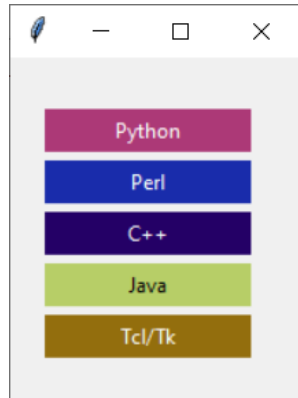
```
import tkinter as tk
import random

root = tk.Tk()
# width x height + x_offset + y_offset:
root.geometry("170x200+30+30")

languages = ['Python', 'Perl', 'C++', 'Java', 'Tcl/Tk']
labels = range(5)
for i in range(5):
    ct = [random.randrange(256) for x in range(3)]
    brightness = int(round(0.299*ct[0] + 0.587*ct[1] + 0.114*ct[2]))
    ct_hex = "%02x%02x%02x" % tuple(ct)
    bg_colour = '#' + "".join(ct_hex)
    l = tk.Label(root,
                  text=languages[i],
                  fg='White' if brightness < 120 else 'Black',
                  bg=bg_colour)
    l.place(x = 20, y = 30 + i*30, width=120, height=25)

root.mainloop()
```


Output:



Grid Manager

The first geometry manager of Tk had been pack. The algorithmic behaviour of pack is not easy to understand and it can be difficult to change an existing design. Grid was introduced in 1996 as an alternative to pack. Though grid is easier to learn and to use and produces nicer layouts, lots of developers keep using pack.

Grid is in many cases the best choice for general use. While pack is sometimes not sufficient for changing details in the layout, place gives you complete control of positioning each element, but this makes it a lot more complex than pack and grid.

The Grid geometry manager places the widgets in a 2-dimensional table, which consists of a number of rows and columns. The position of a widget is defined by a row and a column number. Widgets with the same column number and different row numbers will be above or below each other. Correspondingly, widgets with the same row number but different column numbers will be on the same "line" and will be beside of each other, i.e. to the left or the right.

Using the grid manager means that you create a widget, and use the grid method to tell the manager in which row and column to place them. The size of the grid doesn't have to be defined, because the manager automatically determines the best dimensions for the widgets used.

1. Create a python file named PL8Grid_LastnameFirstname.
2. Type the following code:

```
import tkinter as tk

colours = ['red', 'green', 'orange', 'white', 'yellow', 'blue']

r = 0
for c in colours:
    tk.Label(text=c, relief=tk.RIDGE, width=15).grid(row=r, column=0)
    tk.Entry(bg=c, relief=tk.SUNKEN, width=10).grid(row=r, column=1)
    r = r + 1

tk.mainloop()
```

Output:

