NextGen Academy
Towards fulfilling a million dreams

Python Programming

Module V: GUI Programming in Python (using Tkinter/wxPython/Qt)

June 20, 2023

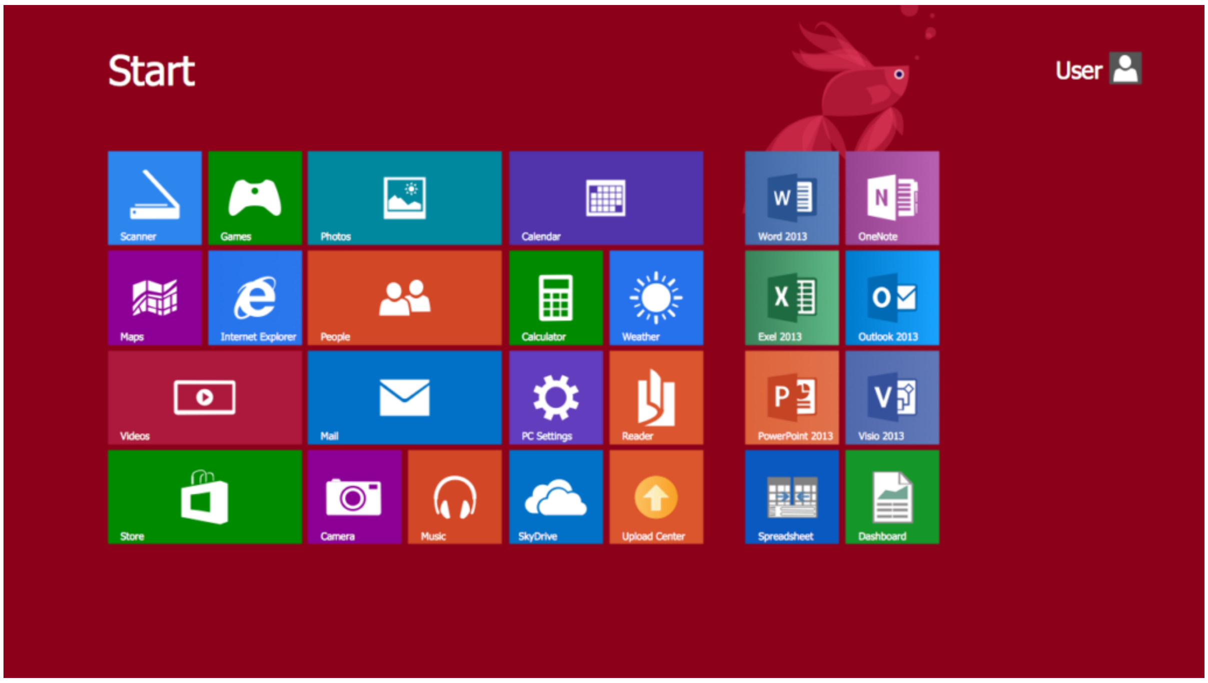
Contents

This unit covers

- Introduction to GUI Programming in Python: In this unit, we embark on a comprehensive exploration of GUI programming using Python, focusing on popular libraries like Tkinter, wxPython, and Qt. The journey begins with an understanding of GUI and its inherent advantages, setting the stage for creating engaging user interfaces.
- GUI Libraries and Layout Management: We delve into the intricacies of GUI libraries, with a special emphasis on layout management. Understanding how to structure and organize elements within a graphical interface is vital for creating visually appealing and intuitive applications.
- Events and Bindings: A pivotal aspect of GUI programming is the management of user interactions. This unit covers events and bindings, providing insights into how to handle user inputs effectively. Mastery of these concepts is crucial for creating responsive and interactive graphical applications.
- Aesthetics in GUI: Fonts, Colors, and Canvas Drawing: The unit explores the visual aspects of GUIs, including the manipulation of fonts and colors to enhance the overall design. Additionally, we delve into the dynamic drawing capabilities offered by the Canvas, allowing us to create and customize graphical elements such as lines, ovals, and rectangles.
- Widget Arsenal: A detailed examination of a diverse set of widgets is undertaken. From fundamental elements like Frames and Labels to interactive components such as Buttons, Check Buttons, Entries, List Boxes, Messages, Radio Buttons, Text, and Spin Boxes, students will gain a comprehensive understanding of the toolkit available for crafting rich and versatile GUIs.
- Practical Application and Hands-On Exercises: The unit is designed to be practical, with hands-on exercises and real-world examples. Students will apply their knowledge to design and implement GUI applications, honing their skills in creating interfaces that not only meet functional requirements but also enhance the overall user experience.

1 What is GUI

Very simply, a Python GUI is a GUI that is written in the Python programming language.



Windows Operating System GUI, Source: OmniSci

Python is a very popular programming language thanks to its great degree of readability, widespread adoption and most importantly, its beginner friendliness. While being incredibly useful for the fields of data science and machine learning, Python is also great for developing graphical user interfaces! In fact, it has many frameworks that even beginners can use to easily get started with developing a GUI.

Thus, if you're starting out with learning Python and want to take it to the next level with GUI programming, this article is for you!

1.1 Python GUI Uses

You can do many things with Python GUI programming – but here are a few of the popular examples!

1.1.1 Building a Mobile Application

With interfaces for users to like, post, comment, or interact in many kinds of ways, mobile applications are some of the best examples of Python GUIs in action! Did you know that some of these popular and successful mobile applications were written in Python? How many of them do you recognise or use yourself?

1.1.2 Games (Flappy Bird, Mount & Blade)

Apart from mobile applications, Python has been used to create some of the games that we know and love! With flashy graphics and rewarding interactivity, games are one use case that heavily leverage GUIs to create value and enjoyment for users. For example, games like Flappy Bird and Mount & Blade were programmed in Python!

2 Advantages of GUI

1. It is user-friendly and simple.

2. It is attractive for non technical users.

3. No need to execute commands for carrying out functions like opening file , saving etc.

4.Look and feel is better than text interface

5. It is much better than command driven interface which has many drawbacks.

6. User can switch quickly between tasks on the GUI interface.

3 Introduction to GUI library

We can use any of the following toolkits in Python for GUI programming.

Tkinter: Tkinter is a standard package used for GUI programming in Python. This is built on top of the Tk interface.

PyQt: PyQt is a Python toolkit binding of the Qt toolkit. Qt is a C++ framework that is used by Python to implement a cross-platform PyQt toolkit as a plug-in.

wxPython: wxPython is also a cross-platform GUI toolkit. It is a wrapper for the API wxWidgets.

4 Layout management

- Pack Layout Manager
- Grid Layout Manager
- Place Layout Manager

4.1 Pack Manager

pack is one of the oldest and most used layout manager in Python's TKInter package. It is very easy to style and place the widgets in an app using this pack() manager. When you use this pack() method on a widget, you don't need to explicitly specify the position of that widget, pack automatically places the widget in the window based on the space available in the window.

You can use pack when your layout only consists of a group of items all aligned horizontally or vertically, mostly in the case of a navigation menu or something like that.

This pack() has 3 options to use they are: fill, expand, and side. I will create a simple example to demonstrate this pack manager.

```
# fill option

label1 = Label(root, text="Label 1", bg="#E74C3C", fg="white").pack(fill=X, padx=10)

label2 = Label(root, text="Label 2", bg="#2ECC71", fg="black").pack(fill=X, padx=10)

label3 = Label(root, text="Label 3", bg="#F1C40F", fg="white").pack(fill=X, padx=10)
```

Use side option to align them horizontally.

```
# side option

label4 = Label(root, text="Label 1", bg="#34495E", fg="white").pack(fill=X, padx=10, pady=10, side=LEFT)

label5 = Label(root, text="Label 2", bg="#5DADE2", fg="black").pack(fill=X, padx=10, side=LEFT)

label6 = Label(root, text="Label 3", bg="#A569BD", fg="white").pack(fill=X, padx=10, side=LEFT)
```

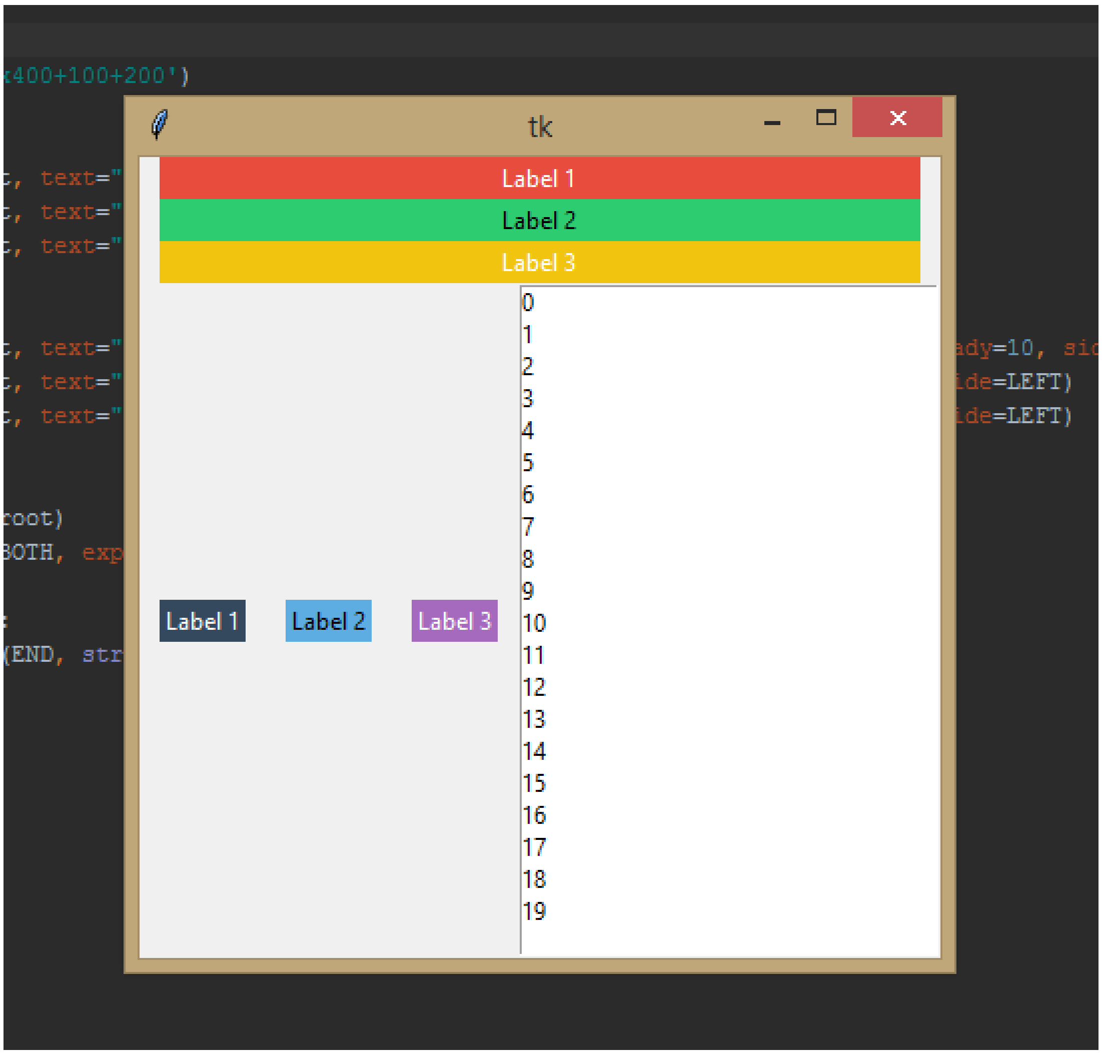
Expand operation to take up the full height until the content ends. We use listbox widget to fill up space.

```
# expand option

listbox = Listbox(root)

listbox.pack(fill=BOTH, expand=1)

for i in range(20):

listbox.insert(END, str(i))
```

4.2 Pack Manager Example:

```
from tkinter import *

root = Tk()

root.geometry('400x400+100+200')
```

```
# fill option

label1 = Label(root, text="Label 1", bg="#E74C3C", fg="white").pack(fill=X, padx=10)

label2 = Label(root, text="Label 2", bg="#2ECC71", fg="black").pack(fill=X, padx=10)

label3 = Label(root, text="Label 3", bg="#F1C40F", fg="white").pack(fill=X, padx=10)

# side option

label4 = Label(root, text="Label 1", bg="#34495E", fg="white").pack(fill=X, padx=10, pady=10, side=LEFT)

label5 = Label(root, text="Label 2", bg="#5DADE2", fg="black").pack(fill=X, padx=10, side=LEFT)

label6 = Label(root, text="Label 3", bg="#A569BD", fg="white").pack(fill=X, padx=10, side=LEFT)

# expand option

listbox = Listbox(root)

listbox.pack(fill=BOTH, expand=1)

for i in range(20):

listbox.insert(END, str(i))

mainloop()
```



```
x400+100+200')

t, text="
t, text="
t, text="

t, text="                                                   ady=10, sid
t, text="                                                   ide=LEFT)
t, text="                                                   ide=LEFT)

root)
BOTH, exp

:
(END, str
```

Layout Managers in Python GUI – Pack, Grid and Place

4.3 Grid Manager

grid is one of the most flexible layout manager out of the three GUI layout managers in Python. It was introduced as an alternative to pack. Grid allows you to position the elements in rows and columns which gives you more flexibility with your widgets.

***from** tkinter **import** **

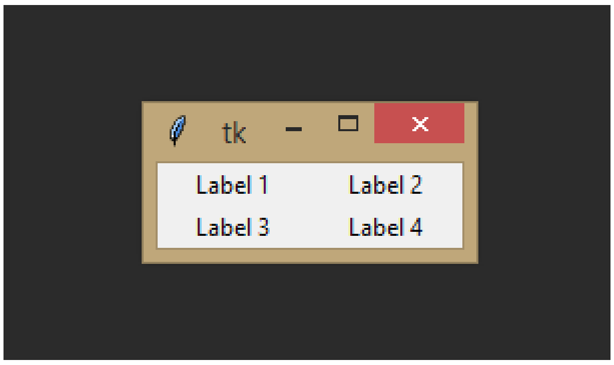
Label(text="Label 1", width=10).grid(row=0, column=0)

Label(text="Label 2", width=10).grid(row=0, column=1)

Label(text="Label 3", width=10).grid(row=1, column=0)

Label(text="Label 4", width=10).grid(row=1, column=1)

mainloop()

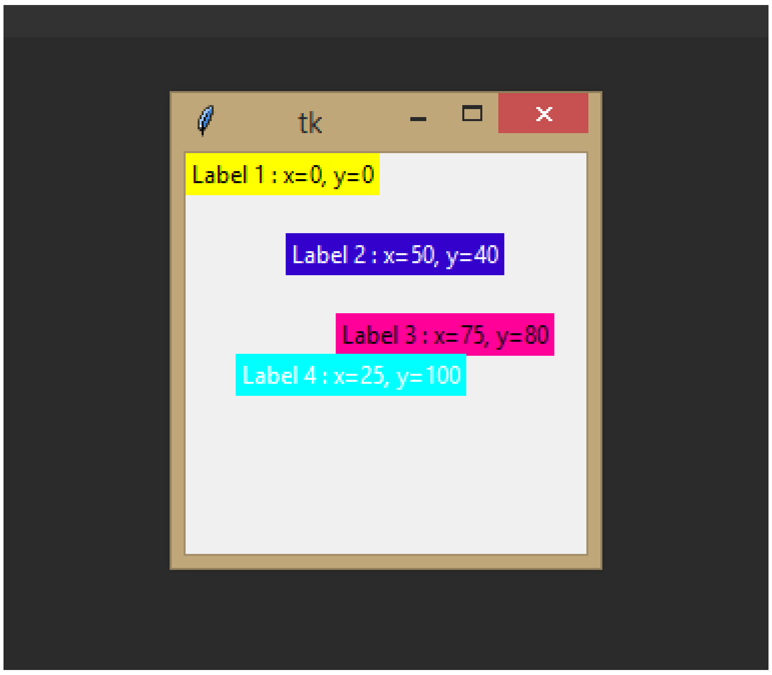


Layout Managers in Python GUI – Pack, Grid and Place

4.4 Place Manager

place is the most complex manager out of the 3 managers. It uses absolute positioning, when you choose place as your layout manager, then you need to specify the widgets positioning using x and y coordinates, When using this layout manager the size and position of the widgets do not change while resizing the window.

```
from tkinter import *

root = Tk()

root.geometry('200x200+100+200')

Label(root, text="Label 1 : x=0, y=0", bg="#FFFF00", fg="black").place(x=0, y=0)

Label(root, text="Label 2 : x=50, y=40", bg="#3300CC", fg="white").place(x=50, y=40)

Label(root, text="Label 3 : x=75, y=80", bg="#FF0099", fg="black").place(x=75, y=80)

Label(root, text="Label 4 : x=25, y=100", bg="#00FFFF", fg="white").place(x=25, y=100)

mainloop()
```



5 Events and bindings

5.1 What is an event?

The basic definition of the word event is the occurrence of a particular thing or could be defined as a thing that happens or takes place similarly in Tkinter events are operations carried out by the user such as mouse movement or key press. Events are used to connect the actions performed by the user to a logic understandable by the application which results in its graphical changes.

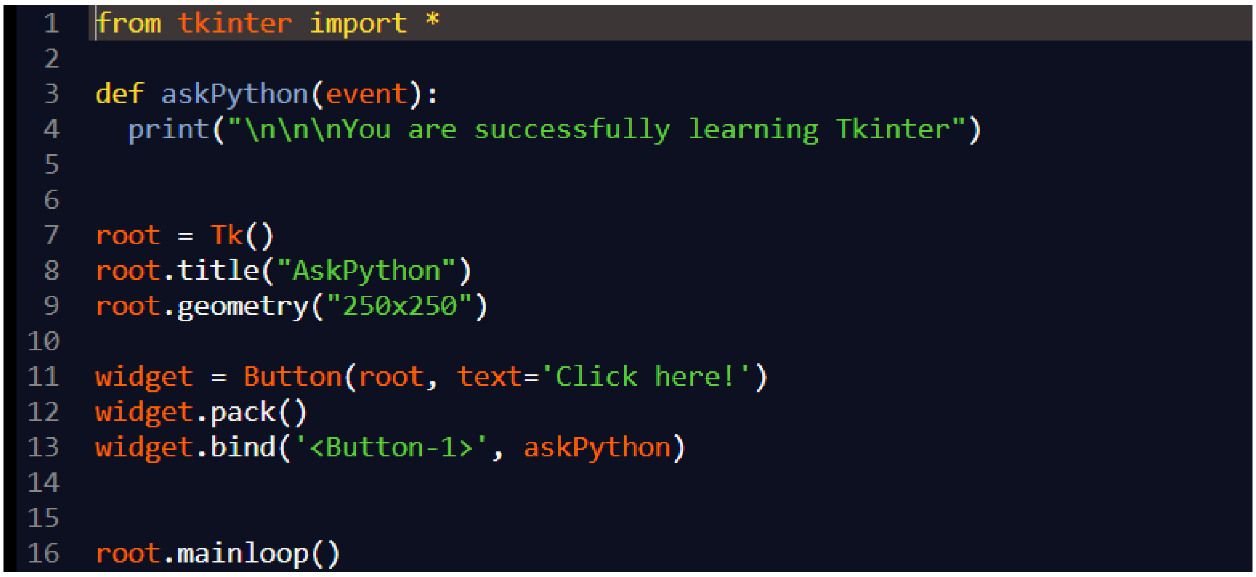
When we use widgets in Tkinter, they do have a lot of built-in behaviours. For instance, a button will execute its command callback in response to a mouse click. However, you can add, modify, or remove actions/behaviour using Tkinter's event-binding functionality.

This binding functionality is achieved by using .bind() function and the event along with the event handler is defined in the widget

widget.bind(event, event handler)

- **Event** – occurrence caused by the user that might reflect changes.
- **Event Handler** – function in your application that gets invoked when the event takes place.
- **Bind** – configuring an event handler (python function) that is called when an event occurs to a widget.

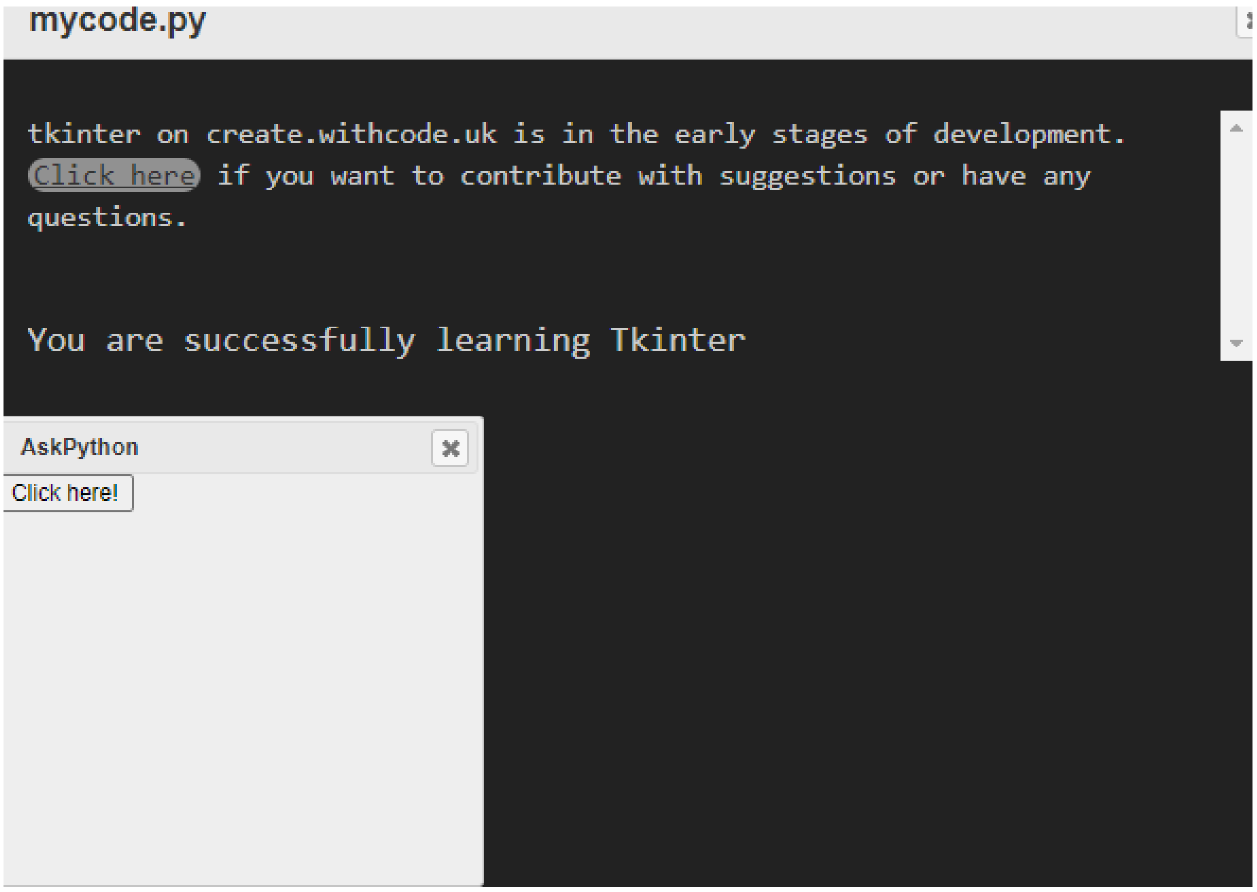
For Example:



```
from tkinter import *

def askPython(event):
  print("\n\n\nYou are successfully learning Tkinter")


root = Tk()
root.title("AskPython")
root.geometry("250x250")

widget = Button(root, text='Click here!')
widget.pack()
widget.bind('<Button-1>', askPython)


root.mainloop()
```

OUTPUT

As an output first a dialogue box will appear with a button displayed on it, once the user clicks the left button on its mouse on the displayed button, the event handler (here the function named askPython acts as an event handler) is invoked and the text message "You are successfully learning Tkinter" will appear in the output window.



```
tkinter on create.withcode.uk is in the early stages of development.
Click here if you want to contribute with suggestions or have any
questions.


You are successfully learning Tkinter
```

Output

5.2 Syntax of the events.

An event sequence is typically a string that contains one or more event patterns. Each event pattern depicts a single possible outcome. If a sequence has many event patterns, the handler won't be called until every pattern occurs in that sequence. Remember event sequence is always described between "<....>"

Syntax:

<modifier - *type* - *detail>*

The modifier and detail parts are optional. The type is the crucial part of defining the event.

- modifier – section to specify combinations, such as pressing and holding the shift or control buttons when using the mouse or clicking other keys.
- type – defines a specific kind of event, like a mouse click or key press.
- detail – elements to use when describing the mouse button or key you're seeking for. This is 1 for button 1, 2 for button 2, or 3 for button 3 on a mouse.

5.3 Types of events

<button>

One of the mouse buttons was clicked by the user. Which button is specified in the detail section Example: <button-1> is the leftmost button, <button-2> is the middle button(if available), <button-3> the rightmost button, <button-4>

for scroll up and <button-5 > for scroll down.

<buttonRelease>

The mouse button was released by the user. Since the user may move the mouse off the widget to prevent mistakenly pressing the button, this is undoubtedly a better alternative in most scenarios than the Button event.

<Configure>

The user modified the size of a widget, for example by extending a window's corner or side.

<Destroy>

A widget is being destroyed.

<Enter>

The user placed the mouse pointer over a widget's visible area. (This is distinct from the enter key, which is a KeyPress event for a key)

<Expose>

This happens whenever at least a portion of your widget or application becomes visible after being hidden by another window.

<FocusIn>

This may occur in response to user interaction (such as using the tab key to switch across widgets) or programmatically (for instance, when your software invokes the .focus set() method on a widget).

<FocusOut>

The widget that was the input focus was changed. Similar to <FocusIn>, either the user or your software can initiate this event.

<KeyPress>

The user pressed a keyboard key. This key is specified in the detail section of the event section. The word "key" can be used as an abbreviation.

<KeyRelease>

The user released the key.

<Leave>

The user moved the mouse pointer out of a widget.

<Map>

An application is mapping, or making a widget visible. For instance, this will take place if you use the widget's.grid() method.

<Motion>

When a mouse button is depressed, the mouse is moved. Use <B1-Motion>, <B2-Motion> or <B3-Motion> to select the left, middle, or right mouse button, respectively.

<MouseWheel>

The user moved the mouse wheel up or down.

<Unmap>

Unmapped widgets are those that are no longer visible. For instance, when you use the widgets.grid remove() method, this occurs.

<Visibility>

When at least a portion of the application window is visible on the screen, this occurs.

6 Font Colors

- **foreground** or **fg** is the option that accepts the color input from the user and sets the font or text color.
- This option is common in almost all the widgets and can be applied on the text of any widget.
- In our example, we will be implementing color on the Text box widget. By default the Text color is blue but we will change it to Blue.

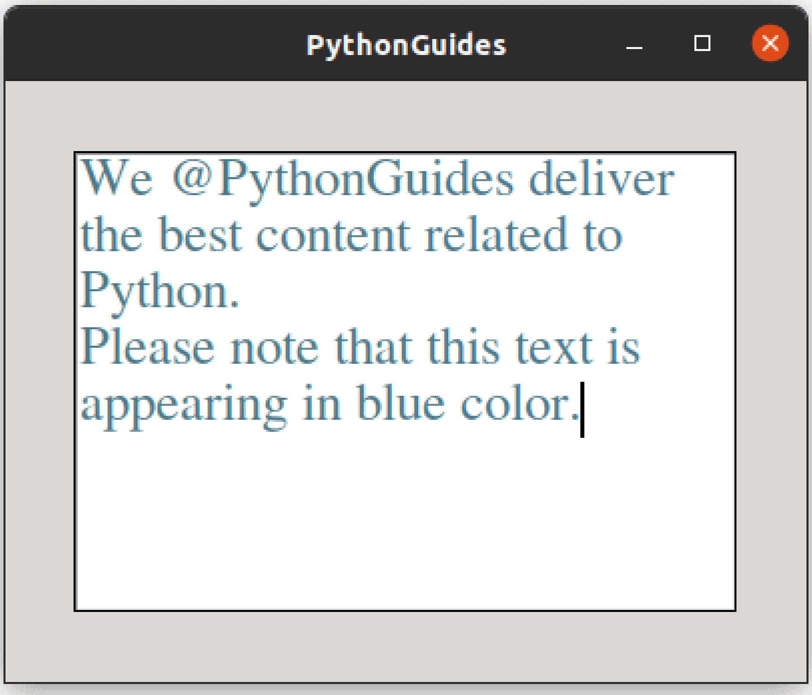
```
from tkinter import *

ws = Tk()

ws.title('PythonGuides')

ws.config(bg='#D9D8D7')

ws.geometry('400x300')

tb = Text(

ws,

width=25,

height=8,

font=('Times', 20),

wrap='word',

fg='#4A7A8C'

)
```

tb.pack(expand=True)

ws.mainloop()

Output of Python Tkinter Color Text

In this output, we have changed the default color of the text from black to light blue.



We @PythonGuides deliver the best content related to Python.
Please note that this text is appearing in blue color.

Python Tkinter Text Color

7 drawing on Canvas (line, oval, rectangle, etc.)

The Canvas element that comes with Tkinter is quite versatile. Out of the box you can draw simple basic shapes like squares and circles, but also lines and more complex shapes made up of points. You can even add text and images to the canvas.

This article will go through all of the different types of items you can draw using the Tkinter Canvas object.

Before getting into that, it's important to understand how coordinates are used on a canvas object. Drawing items on a canvas requires the use of an x and y coordinate to pinpoint where the item is to be drawn. All points are relative to the top left hand corner, so a coordinate of 0,0 would be right in the top left corner.

Because it is possible to create a canvas within a scrollable area, the canvas might be offset from the current window view. In this case the coordinates are still measured from the top left hand corner of where the canvas is visible in the current window. In this case a coordinate of 0,0 would be in the top left hand corner of the application window.

This article will cover the following types of canvas element:

- Line

- Rectangle
- Oval
- Arc
- Polygon
- Bitmap
- Image
- Text
- Window

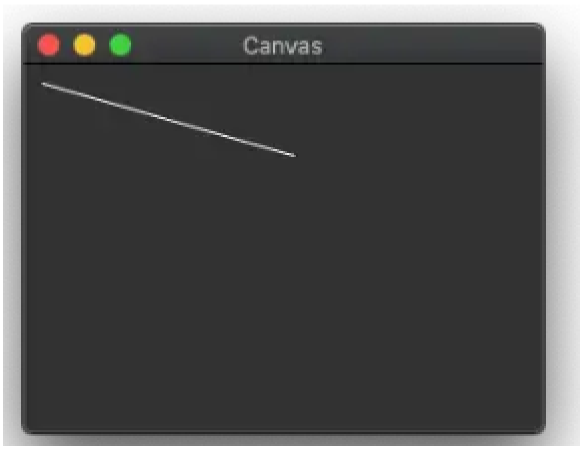
7.1 Line

To draw a line we use the create_line() method. This takes a series of x and y coordinates to draw the line. As a minimum you need to supply 2 coordinates for the start and end of the line.

In the example below the line is drawn from the coordinates 10,10 to 150,50 on the canvas.

```
import tkinter as tk

from tkinter import Canvas

app = tk.Tk()

app.title("Canvas")

canvas = Canvas(app)

canvas.pack()

canvas.create_line(10, 10, 150, 50)

app.mainloop()
```

This creates the following application.



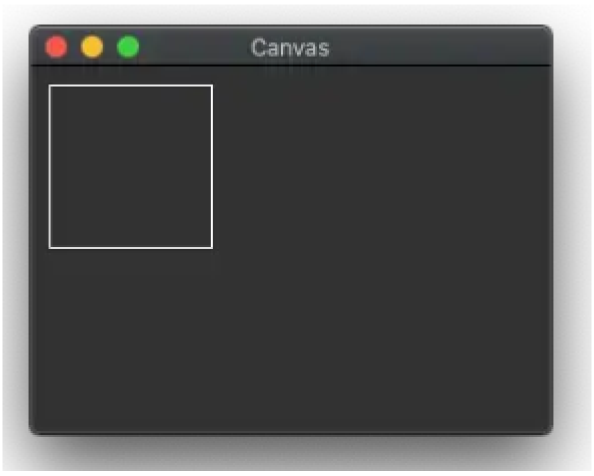
The points sent to the create_line() method can be passed as an array of points. The following will produce exactly the same line as above.

```
points = [
10, 10,
150, 50
]
canvas.create_line(points)
```

You can keep adding points to the line method to create more parts of the line. Adding a 3rd point will create a line with two parts, adding a 4th point will create a line with three parts. For example, the following will draw a box in the upper left hand side of the canvas element.

```
canvas.create_line(10, 10, 10, 100, 100, 100, 100, 10, 10, 10)
```

Running this set of coordinates produces a canvas that looks like this.



An alternative way of writing this code is to pass the points to the create_line() method as a single variable. This will produce the same shape as the above application.

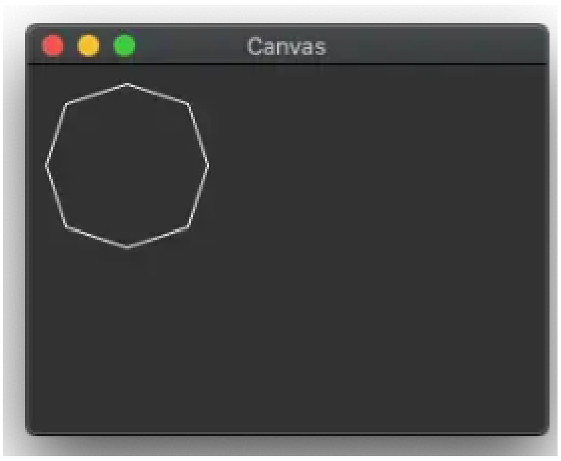
```
points = [
10, 10, 10, 100, 100, 100, 100, 10, 10, 10
]
canvas.create_line(points)
```

Instead of a straight line you can supply the *smooth* configuration option, which will draw the line as a series of parabolic lines rendered in straight line segments. The *splinesteps* option controls how many segments each of the lines with have. In the example below the splinesteps is set to 2, which means that each line is smoothed with 2 lines.

```
points = [
10, 10, 10, 100, 100, 100, 100, 10, 10, 10
]
```

canvas.create_line(points, smooth='true', splinesteps=2)

This produces the following application. The 8 sided shape is really the 4 lines drawn in the previous application but with a smooth operation added to the lines.



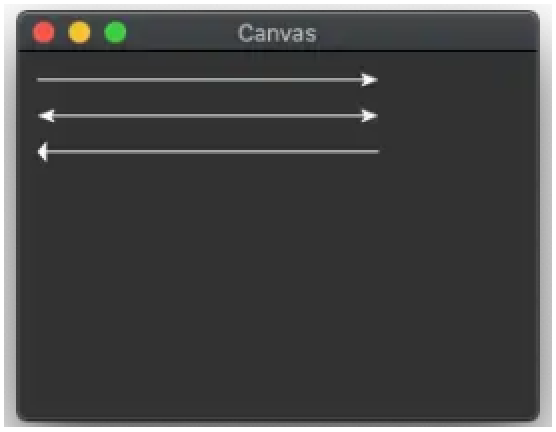
It is also possible to add arrows to lines using the *arrow* configuration option. Using tk.LAST adds the arrow to the last point drawn, tk.FIRST to the first point drawn, and tk.BOTH to add the arrow to both ends. You can also use the *arrowshape* config option set an arrow shape using an array of three items that describe the shape of the arrow head.

canvas.create_line(10, 15, 200, 15, arrow=tk.LAST)

canvas.create_line(10, 35, 200, 35, arrow=tk.BOTH)

canvas.create_line(10, 55, 200, 55, arrow=tk.FIRST, arrowshape=[5,5,5])

These three lines will produce the following application.



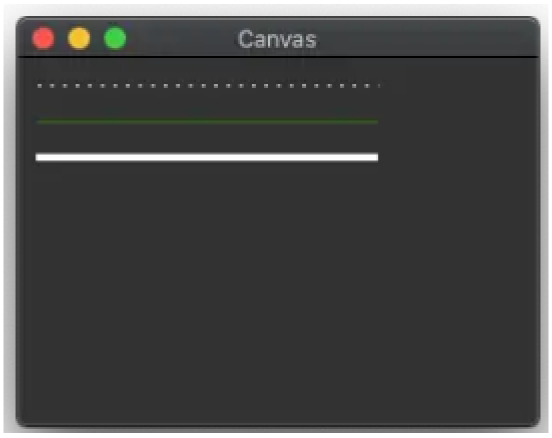
The canvas line is actually quite a good starting point to drawing on a Canvas as there are quite a lot of options available to it and some of these options are available to other objects. For example, we could create a dashed line, a different coloured line, or even a thick line by setting different configuration options.

canvas.create_line(10, 15, 200, 15, dash=[2,5])

canvas.create_line(10, 35, 200, 35, fill='green')

canvas.create_line(10, 55, 200, 55, width=4)

These options will produce the following application.



To draw a singe pixel, which is sometimes needed, you can use the create_line method to draw a point at a single location with a length of 1 pixel.

```
x = 10

y = 10

canvas.create_line(x, y, x + 1, y)
```

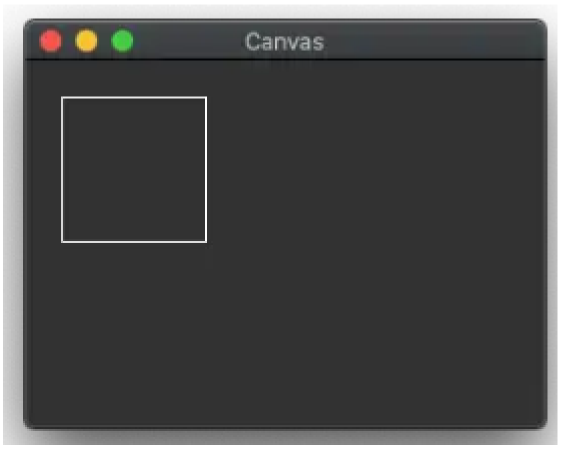
Whilst the arrow remains a part of the canvas line, the dash, fill and width of the line can be controlled across most of the different line canvas elements.

7.2 Rectangle

The create_rectangle() method creates a rectangle shape, essentially creating any four sided regular shape. By default, the two coordinates given are the top left and bottom right of the rectangle produced.

```
import tkinter as tk

from tkinter import Canvas

app = tk.Tk()

app.title("Canvas")

canvas = Canvas(app)

canvas.pack()

canvas.create_rectangle(20, 20, 100, 100)

app.mainloop()
```

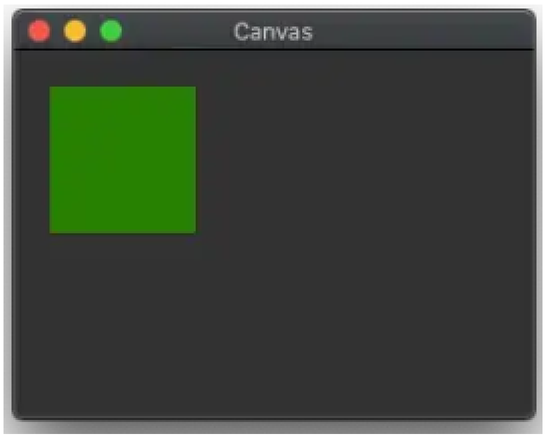
This creates the following application.



The main difference between the line and the rectangle is that the *fill* configuration item will fill in the inner portion of the shape, instead of colouring the line. The line can be coloured using the *outline* configuration option. In order to fully colour the rectangle you can use the following code. You can also set the *outline* to be an empty string to set it to be invisible.

canvas.create_rectangle(20, 20, 100, 100, fill='green', outline='green')

This will produce the following application.



7.3 Oval

The create_oval() method will create an oval, but will create a circle if given equal coordinates. The oval will be drawn between the top left and bottom right coordinates. If the difference between the top to bottom and left to right is the same then a circle will be drawn.

For example, if the used the same dimensions to draw a square, but instead use the create_oval() method then a circle will be drawn.

import tkinter as tk

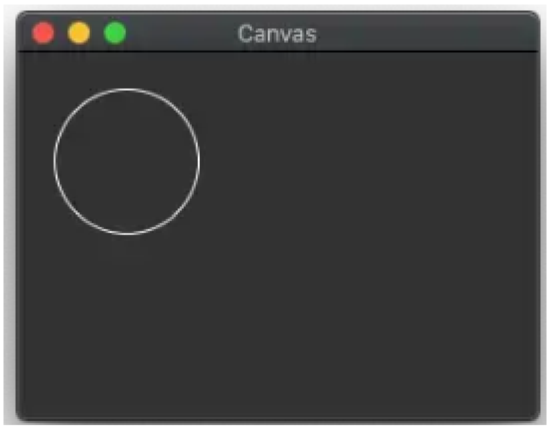
from tkinter import Canvas

app = tk.Tk()

app.title("Canvas")

```
canvas = Canvas(app)

canvas.pack()

canvas.create_oval(20, 20, 100, 100)

app.mainloop()
```

This creates the following application.

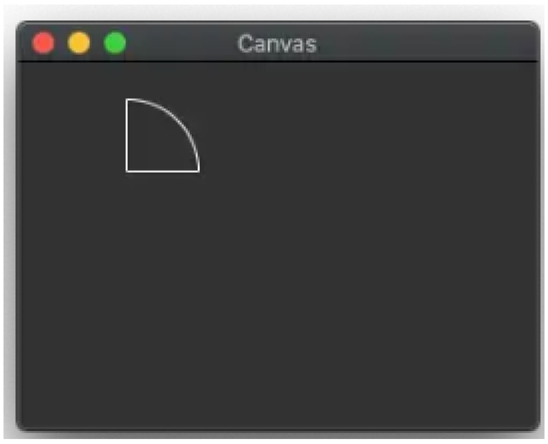


7.4 Arc

The create_arc() method will create a segment of a circle. This follows the same rules as the create_oval() method, but in this case the coordinates will only draw a part of the oval of the same dimensions.

```
import tkinter as tk

from tkinter import Canvas

app = tk.Tk()

app.title("Canvas")

canvas = Canvas(app)

canvas.pack()

canvas.create_arc(20, 20, 100, 100)

app.mainloop()
```

This creates the following application.



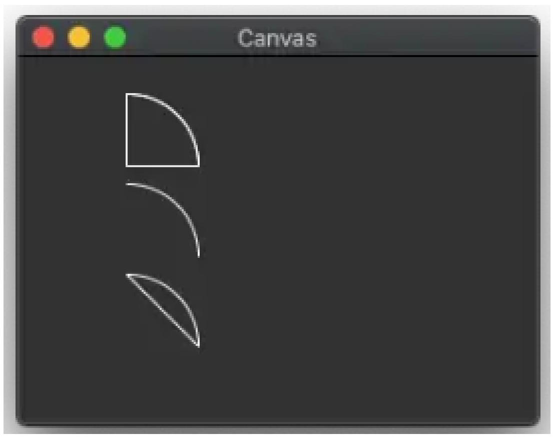
By default, this creates an arc that looks like a slice of pie, this can be controlled through the *style* configuration option, the default being the tk.PIESLICE option. Other options available are tk.ARC and tk.CHORD. These lines show three different arcs being created using the different styles available.

canvas.create_arc(20, 20, 100, 100, style=tk.PIESLICE)

canvas.create_arc(20, 70, 100, 150, style=tk.ARC)

canvas.create_arc(20, 120, 100, 200, style=tk.CHORD)

This creates the following application.



By default, an arc of 90 degrees is drawn from the top of the circle (or 0 degrees). This can be changed using the start and extent configuration options for the create_arc() method.

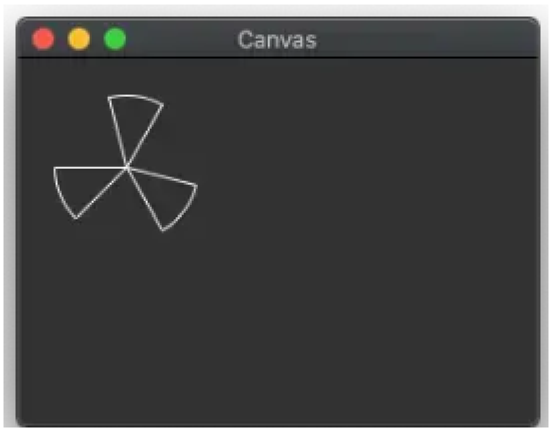
As an example, we can draw three parts of the same circle using the create_acr() method but offset them from each other by 120 degrees.

canvas.create_arc(20, 20, 100, 100, extent=45, start=60)

canvas.create_arc(20, 20, 100, 100, extent=45, start=180)

canvas.create_arc(20, 20, 100, 100, extent=45, start=300)

This draws the following.



7.5 Polygon

The create_polygon() method allows you to create any shape and will fill in the middle part of the polygon. Like the create_line() method, this will accept any number of points that will form the shape.

The following application will create a polygon with 10 random points, which will create shapes of all sorts of size.

```
import tkinter as tk

from tkinter import Canvas

import random

app = tk.Tk()

app.title("Canvas")

canvas = Canvas(app)

canvas.pack()

points = [

random.randrange(0, 200),

random.randrange(0, 200),

random.randrange(0, 200),

random.randrange(0, 200),

random.randrange(0, 200),

random.randrange(0, 200),

random.randrange(0, 200),

random.randrange(0, 200),

random.randrange(0, 200),
```

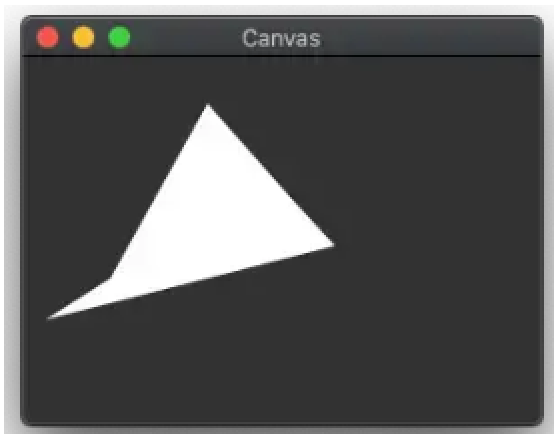
random.randrange(0, 200)

]

canvas.create_polygon(points)

app.mainloop()

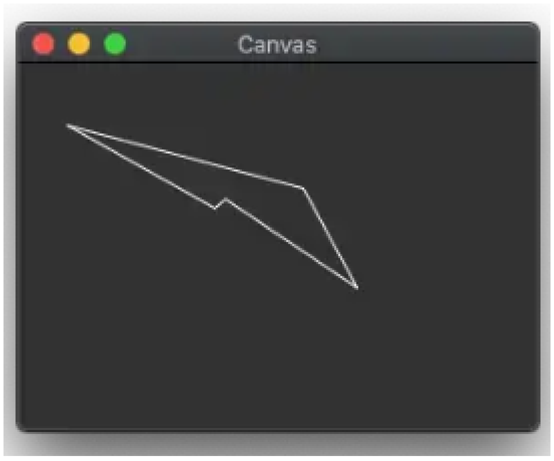
This creates the following application.



The polygon shape can be shown as an outline by setting *fill* to a blank string and setting the *outline* to a colour.

canvas.create_polygon(points, fill='', outline='white')

Using the same random points as above this generates the following application.



7.6 Bitmap

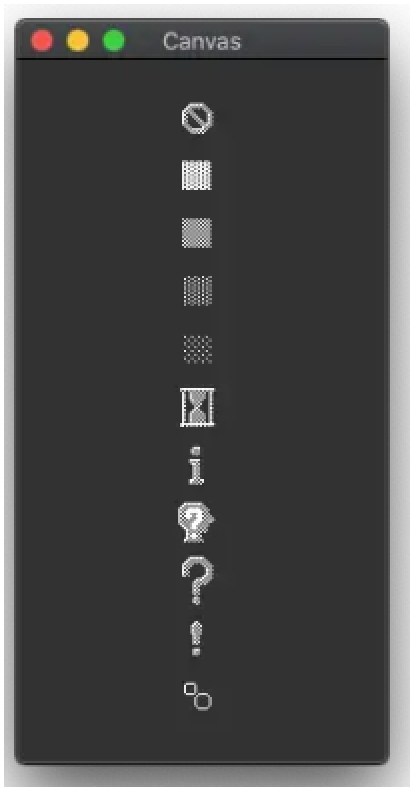
The create_bitmap() method will print out a little icon image. There are 10 different built in bitmaps to pick from, but you can also load in your own xbm files to the function.

The following code creates a canvas object that contains all 10 of the built in bitmap images, but also adds in a custom bitmap called circles.xbm. The custom file is loaded in using a @ symbol in front of the filename. The file referenced here is in the same directory as the script.

import tkinter as tk

```
from tkinter import Canvas

app = tk.Tk()

app.title("Canvas")

bitmaps = [

"error",

"gray75",

"gray50",

"gray25",

"gray12",

"hourglass",

"info",

"questhead",

"question",

"warning",

"@circles.xbm"

]

canvas = Canvas(app, width=200, height=len(bitmaps) * 32 + 32)

canvas.pack()

for i in range(0, len(bitmaps)):

canvas.create_bitmap(100, ((i+1) * 32), bitmap=bitmaps[i], anchor=tk.CENTER)

app.mainloop()
```

This creates the following application.



The final image (with the two circles) is the custom circles.xbm file.

7.7 Image

Often useful in Canvas elements is the inclusion of images. This is made available using the create_image() method.

The method takes the coordinates of the image and a reference to the image object. The image object must be created by passing the file into a PhotoImage() class.

An important consideration of images is where the anchor lies. By default, images are anchored to the middle of the image. This means that the coordinates you enter need to be offset by the height and width of the image. Using the *anchor* configuration setting we can set this to one of the corners of the image.

In the example below we create logo as a PhotoImage object, which we then place at the coordinates 20,20 and use the anchor configuration to set the anchor to the top left hand side. As this is the same position that we are trying to place the image it is pinned to the coordinates 20,20 in the top left hand side.

import tkinter as tk

from tkinter import Canvas, PhotoImage

app = tk.Tk()

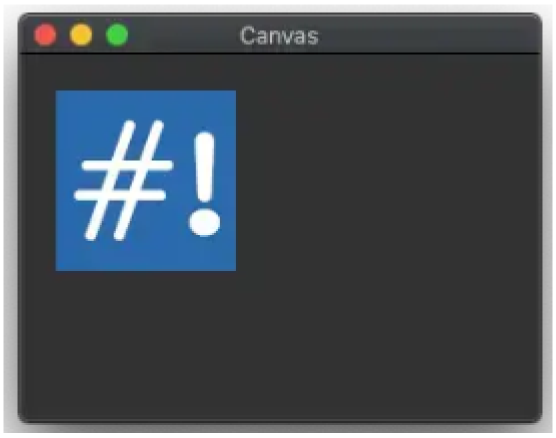
app.title("Canvas")

canvas = Canvas(app)

canvas.pack()

```
logo = PhotoImage(file='hashbangcode_logo.png')

canvas.create_image(20, 20, image=logo, anchor=tk.NW)

app.mainloop()
```

This creates the following application.

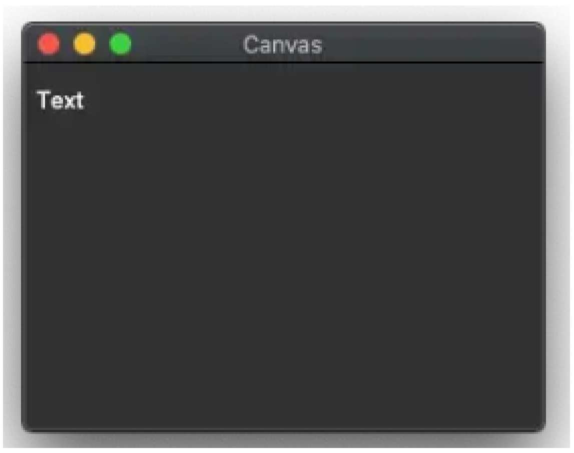


7.8 Text

Finally, it is also possible to add text to your application using the create_text() method. This takes the coordinates that the text must be placed at, and the text itself in the form of a configuration value.

```
import tkinter as tk

from tkinter import Canvas

app = tk.Tk()

app.title("Canvas")

canvas = Canvas(app)

canvas.pack()

canvas.create_text(20, 20, text='Text')

app.mainloop()
```

This creates the following application.



7.9 Window

Finally, it is possible to add other Tkinter interface elements to a canvas using the create_window() method. This method accepts a position and a Tkinter element and will render that element into the Canvas area. The rendered element will always be on top of any canvas operations you perform.

The following code will create a Tkinter canvas element that contains a button. Clicking the button will run the draw_line function that will then draw a random line on the canvas.

```
import tkinter as tk

from tkinter import Canvas

import random

app = tk.Tk()

app.title("Canvas")

app.geometry("200x200")

def draw_line():

points = [

random.randrange(-10, 200),

random.randrange(-10, 200),

random.randrange(-10, 200),

random.randrange(-10, 200)

]

canvas.create_line(points)

canvas = Canvas(app, width=200, height=200)
```

canvas.pack()

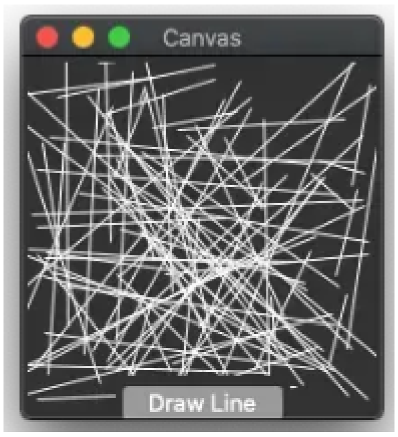
widget = tk.Button(canvas, text='Draw Line', command=draw_line)

widget.pack()

canvas.create_window(100, 190, window=widget)

app.mainloop()

After running this application, and clicking the "Draw Line" button a few times we see this.



The create_window() method is useful for drawing user interface elements on a Canvas and because you can't drawn on top of the window element you can build interfaces on top of your Canvas without them being drawn over.

8 Widget such as : Frame, Label Button Check button, Entry, List box, Message, Radio button, Text, Spin box etc .

As said before we have 19 widgets in the Tkinter library which include text fields, labels, buttons, etc. Before discussing the widgets let us use functions offered by Tkinter to organize these widgets.

1. pack(): This method organizes widgets in blocks before positioning the parent widget

2. grid(): This method organizes widgets in the grid or table form before positioning the parent widget

3. place(): This method organizes the widgets by placing them in the position mentioned in the code.

Now let us see each of these widgets:

8.1 Button:

To add a button to the window, we use the Button() method. Its syntax is given below.

btn=Button(master, option=value)

The parent window is given as the master. Besides this, we have many other parameters that set the features of the button. These include:

a. activebackground: This parameter sets the background color when the cursor is on the button

b. activeforeground: This parameter sets the foreground color when the cursor is on the button

c. bg: We give the background color of the button to the parameter

d. command: We give a function to this parameter. This function contains the code to set the functionality of the button.

e. font: This parameter sets the font style of the label of the button

f. image: This parameter sets the background of the button to the image given

g. width: This sets the width of the button to the given value

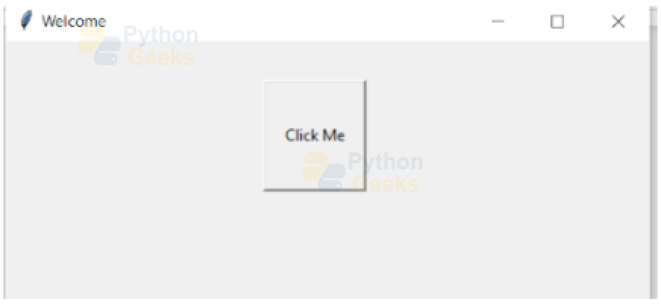
h. height: This sets the height of the button to the given value

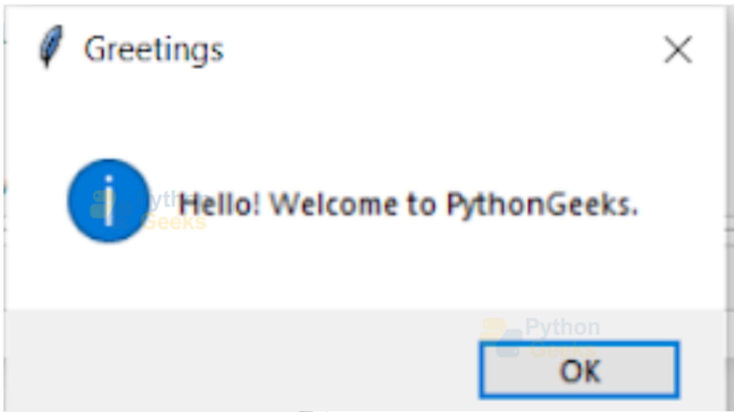
Now let us see an example of creating a button.

Example of the button

```
import tkinter.messagebox

from tkinter import *

win=Tk() #creating the main window and storing the window object in 'win'

win.title('Welcome') #setting title of the window

win.geometry('500x200') #setting the size of the window

def func():#function of the button

tkinter.messagebox.showinfo("Greetings","Hello! Welcome to PythonGeeks.")

btn=Button(win,text="Click Me", width=10,height=5,command=func)

btn.place(x=200,y=30)

win.mainloop() #running the loop that works as a trigger
```

Output:





Hello! Welcome to PythonGeeks.

8.2 Canvas:

This lets you draw shapes such as lines, polygons, etc., and other layouts like graphics, text, and widgets. Its syntax is given below.

can=Canvas(master, option=value)

The parent window is given as the parameter master. In addition, we have parameters like:

a. bd: This parameter sets the border width. The value should be given in pixels

b. bg: We give the background color of the canvas to this parameter

c. cursor: This sets the cursor used when the cursor is on the canvas

d. highlightcolor: This parameter sets the color of the focus highlight

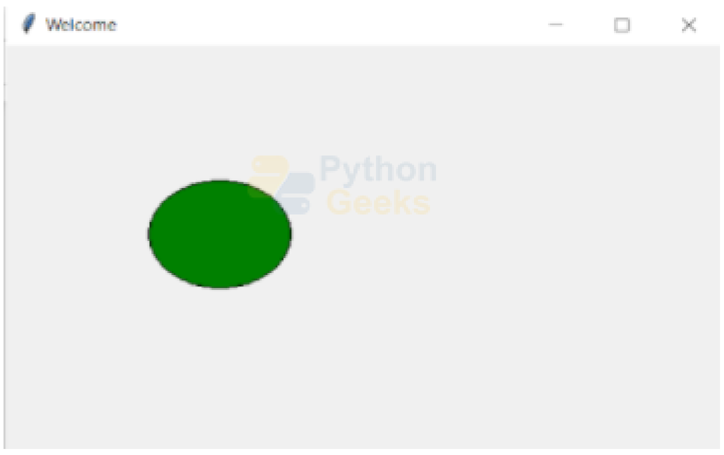
e. width: This sets the width of the canvas to the given value

f. height: This sets the height of the canvas to the given value

Example of the canvas

```
import tkinter.messagebox

from tkinter import *

win=Tk() #creating the main window and storing the window object in 'win'

win.title('Welcome') #setting title of the window

can=Canvas(win, width=500, height=300) #creating the canvas

oval=can.create_oval(100, 100, 200, 180,fill='green') #drawing an oval

can.pack()

win.mainloop() #running the loop that works as a trigger
```

Output:



8.3 CheckButton:

This widget lets the display multiple options for the user to select. The user can select any number of checkboxes. Its syntax is:

cb=CheckButton(master, option=value)

The parent window is given as the master. There are many other parameters related to the check button like:

a. activebackground: This parameter sets the background color when the cursor is on the check button

b. activeforeground: This parameter sets the foreground color when the cursor is on the check button

c. bg: We give the background color of the check button

d. command: This takes a function that contains the code to set the functionality of the check button.

e. font: This parameter sets the font style of the label of the button

f. image: This sets the input image on the check button

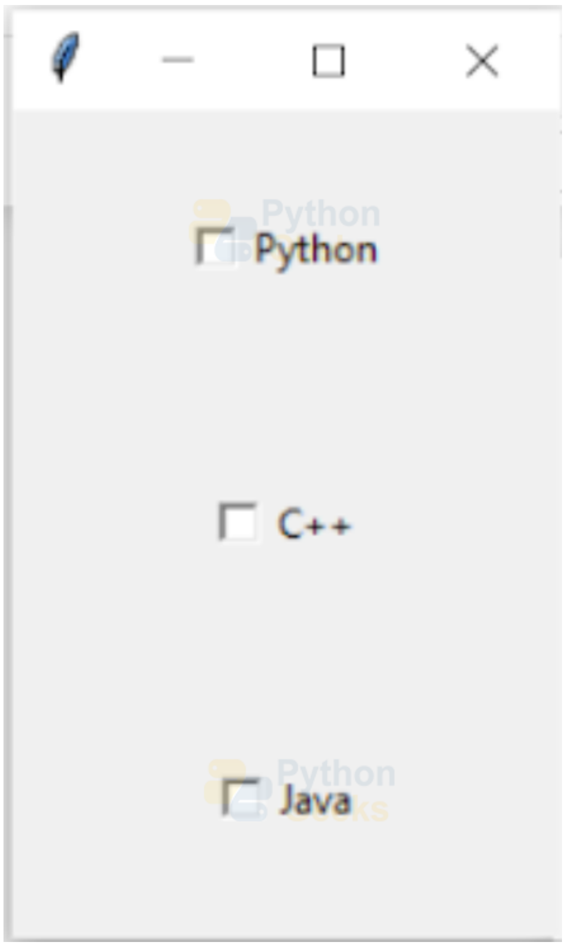
Example of the check button:

```
from tkinter import *

win=Tk() #creating the main window and storing the window object in 'win'

win.title('Welcome') #setting title of the window

cb_var1 = IntVar()

cb1=Checkbutton(win, text='Python', variable=cb_var1,onvalue=1,offvalue=0,height=5,width=20).grid(row=0, sticky=W)

cb_var2 = IntVar()

cb2=Checkbutton(win, text='C++', variable=cb_var2,onvalue=1,offvalue=0,height=5,width=20).grid(row=1, sticky=W)

cb_var3 = IntVar()
```

cb3=Checkbutton(win, text='Java', variable=cb_var3,onvalue=1,offvalue=0,height=5,width=20).grid(row=2, sticky=W)

win.mainloop() #running the loop that works as a trigger

Output:



8.4 Entry:

This widget allows you to take single-line input from the user. Its syntax is:

ent=Entry(master, option=value)

The parent window is given as the master. Besides, it has many other parameters like:

a. bd: This parameter sets the border width in pixels

b. bg: We can set the background color of the entry using this parameter

c. cursor: This sets the cursor used when the cursor is on the entry

d. command: This takes a function as an argument that gets called.

e. highlightcolor: This parameter sets the color of the focus highlight

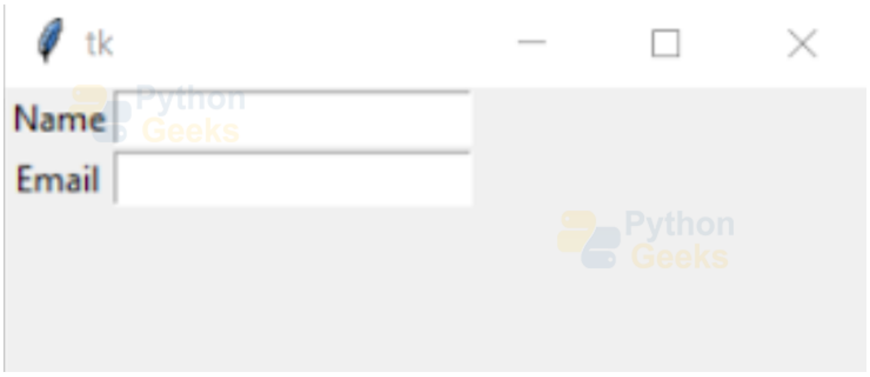
f. width: This sets the width of the entry to the given value

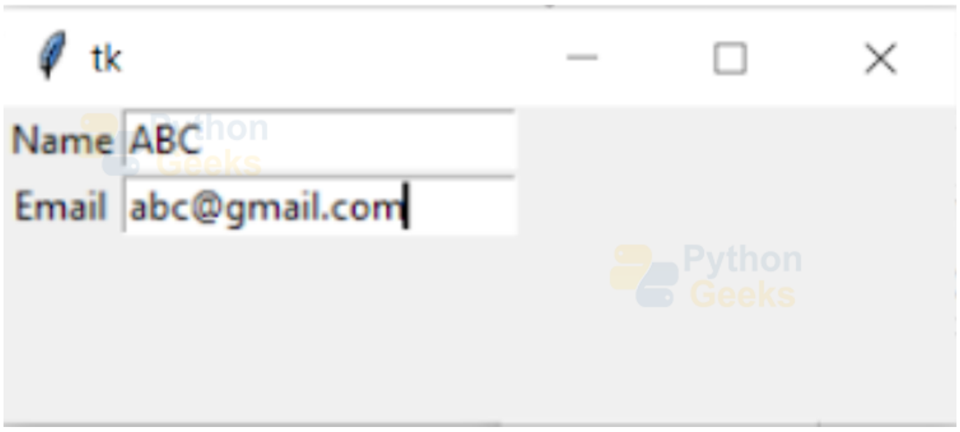
g. height: This sets the height of the entry to the given value

Example of the entry:

```
from tkinter import *

win=Tk() #creating the main window and storing the window object in 'win'

win.geometry('300x100') #setting the size of the window

Label(win, text='Name').grid(row=0)

Label(win, text='Email').grid(row=1)

ent1 = Entry(win)

ent2 = Entry(win)

ent1.grid(row=0, column=1)

ent2.grid(row=1, column=1)

mainloop()
```

Output:





8.5 Frame:

This widget acts as a container of other widgets. This is used to organize and position the widgets. Its syntax is:

frame=Frame(master, option=value)

The parent window is given as the 'master' parameter. There are other parameters like:

a. bd: This parameter sets the border width in pixels

b. bg: We can set the background color of the frame using this parameter

c. cursor: This sets the cursor used when the cursor is on the frame

d. highlightcolor: This parameter sets the color of the focus highlight

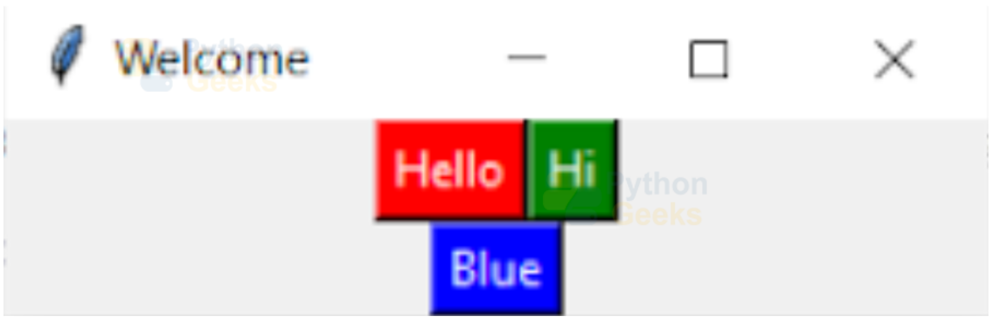
e. width: This sets the width of the frame

f. height: This sets the height of the frame

Example of the frame:

```
from tkinter import *

win=Tk() #creating the main window and storing the window object in 'win'

win.title('Welcome') #setting title of the window

win.geometry('250x50')

frame = Frame(win)

frame.pack()

second_frame = Frame(win)

second_frame.pack( side = BOTTOM )

red_button = Button(frame, text = 'Hello', bg ='red',fg='white')

red_button.pack( side = LEFT)

green_button = Button(frame, text = 'Hi', fg='white',bg='green')

green_button.pack( side = LEFT )

blue_button = Button(second_frame, text ='Blue', fg ='white',bg='blue')

blue_button.pack( side = LEFT )

win.mainloop() #running the loop that works as a trigger
```

Output:



8.6 Label:

This widget lets us display a text or an image. Its syntax is:

lab=Label(master, option=value)

The parent window is given as the 'master' parameter. There are other parameters like:

a. bd: This parameter sets the border width in pixels

b. bg: We can set the background color of the label using this parameter

c. command: This takes a function that contains the code to set the functionality of the label.

d. font: This parameter sets the font style of the text in the label

e. image: This parameter sets the image to display

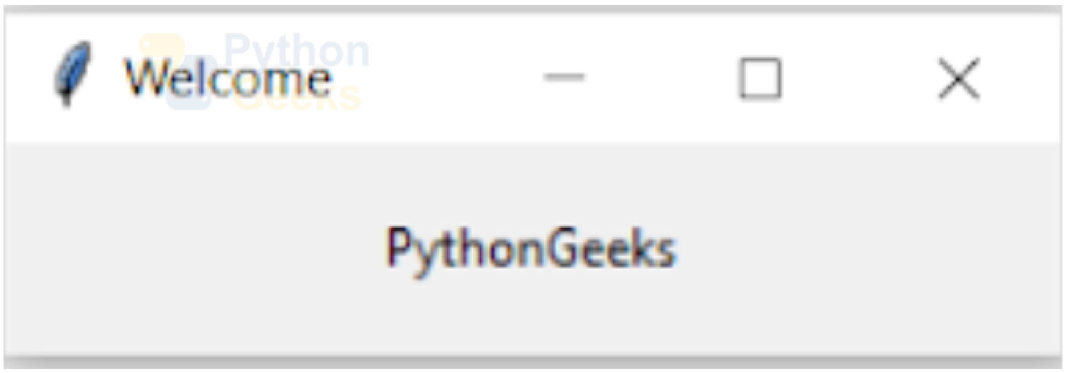
f. width: This sets the width of the label

g. height: This sets the height of the label

Example of the label:

```
from tkinter import *

win=Tk() #creating the main window and storing the window object in 'win'

win.title('Welcome') #setting title of the window

win.geometry('250x50')

lab=Label(win,text='PythonGeeks',width=50,height=30)

lab.pack()

win.mainloop() #running the loop that works as a trigger
```

Output:



8.7 Listbox:

This widget lets us give a list of items to the user to choose from. Its syntax is:

```
lb=Listbox(master, option=value)
```

The parent window is given as the ‘master’ parameter. There are other parameters like:

a. bd: This parameter sets the border width in pixels

b. bg: We can set the background color of the Listbox using this parameter

c. font: This parameter sets the font style of the text of items in the Listbox

d. highlightcolor: This parameter sets the color of the focus highlight

e. image: This parameter sets the background image of the Listbox

f. width: This sets the width of the listbox

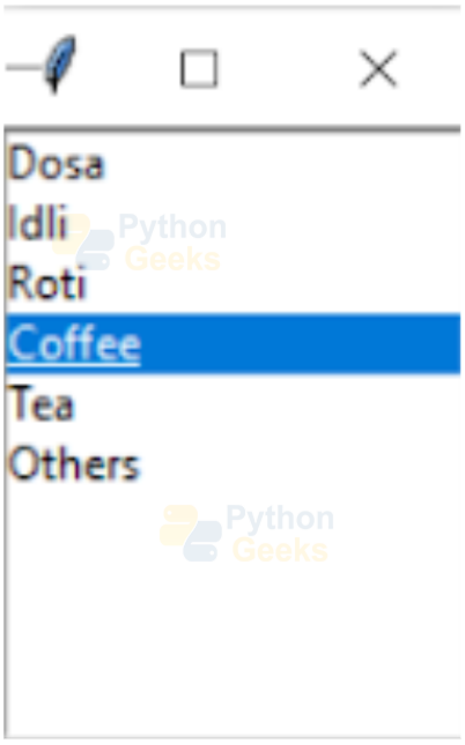
g. height: This sets the height of the Listbox

We can use the insert() function to add elements to the Listbox. For example,

Example of the Listbox:

```
from tkinter import *

win=Tk() #creating the main window and storing the window object in 'win'

win.title('Welcome') #setting title of the window

lb = Listbox(win)

lb.insert(1, 'Dosa')

lb.insert(2, 'Idli')

lb.insert(3, 'Roti')

lb.insert(4,'Coffee')

lb.insert(5,'Tea')

lb.insert(6, 'Others')

lb.pack()

win.mainloop() #running the loop that works as a trigger
```

Output:



8.8 MenuButton:

This lets us add menus to the GUI. We can add a function to the menu button to open the menu options. Its syntax is:

mb=MenuButton(master, option=value)

The parent window is given as the 'master' parameter. Other parameters include:

A. activebackground: This parameter sets the background color when the cursor is on the menu button

B. activeforeground: This parameter sets the foreground color when the cursor is on the menu button

C. bd: This parameter sets the border width in pixels

D. bg: We can set the background color of the menu button using this parameter

E. cursor: This sets the cursor used in the menu button

F. highlightcolor: This parameter sets the color of the focus highlight

G. image: This parameter sets the background image of the menu button

H. width: This sets the width of the menu button

I. height: This sets the height of the menu button

Let us see an example.

Example of the menu button:

*from tkinter import **

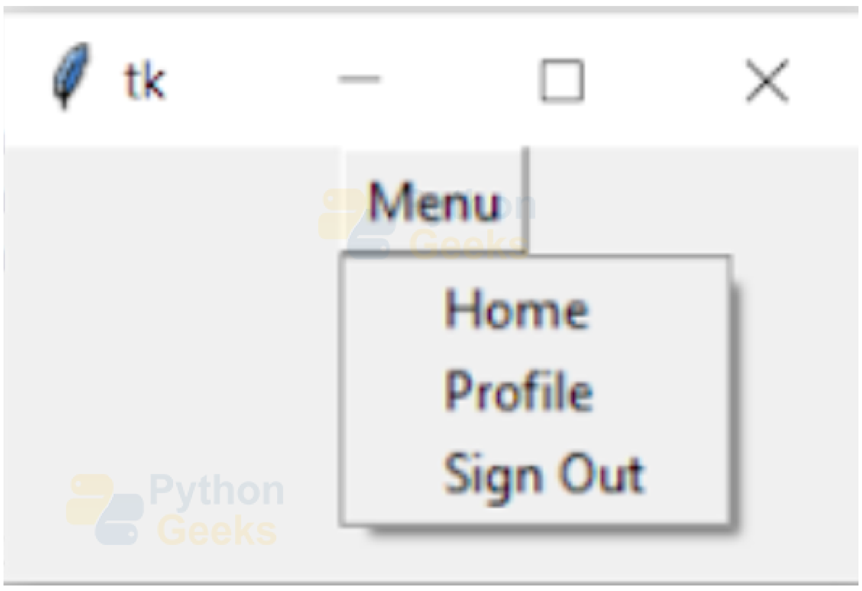
win=Tk() #creating the main window and storing the window object in 'win'

win.geometry('200x100')

mb = Menubutton (win, text = 'Menu')

```
mb.grid()

mb.menu = Menu ( mb, tearoff = 0 )

mb['menu'] = mb.menu

var1 = IntVar()

var2 = IntVar()

var3 = IntVar()

mb.menu.add_checkbutton ( label ='Home', variable = var1 )

mb.menu.add_checkbutton ( label = 'Profile', variable = var2 )

mb.menu.add_checkbutton ( label = 'Sign Out', variable = var3 )

mb.pack()

win.mainloop() #running the loop that works as a trigger
```

Output:



8.9 Menu:

This widget allows us to create all kinds of menus in the application. Its syntax is:

mn=Menu(master, option=value)

The parent window is given as the 'master' parameter. In addition, we can set other parameters like:

A. activebackground: This parameter sets the background color when the cursor is on the menu

B. activeforeground: This parameter sets the foreground color when the cursor is on the menu

C. bg: We can set the background color of the menu using this parameter

D. command: This takes a function that contains the code to set the functionality of the menu

E. font: This parameter sets the font style of the text of items in the menu

F. image: This parameter sets the background image of the menu

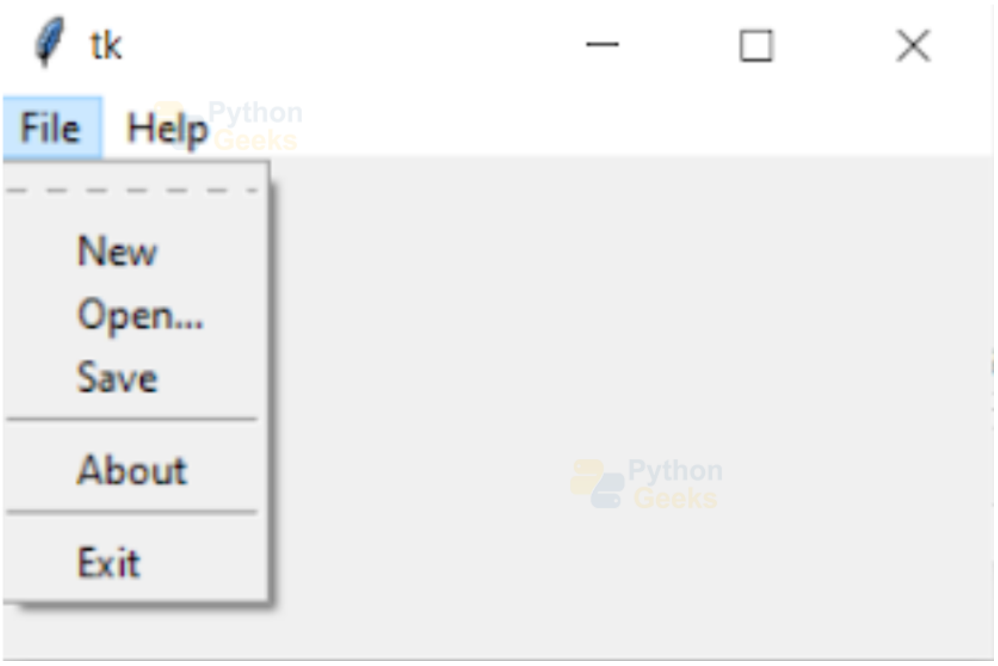
G. title: This sets the title of the menu

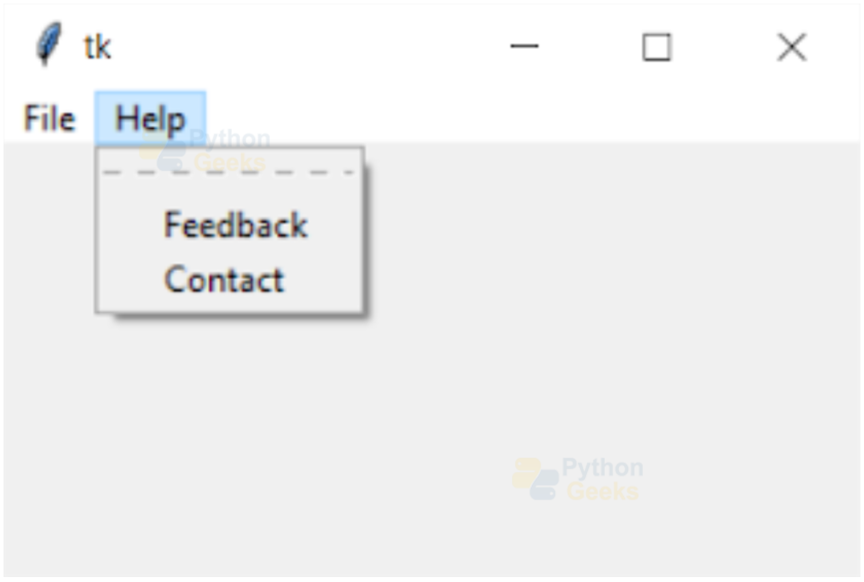
Let us see an example.

Example of the menu:

```
from tkinter import *

win=Tk() #creating the main window and storing the window object in 'win'

win.geometry('300x150')

mn = Menu(win)

win.config(menu=mn)

file_menu = Menu(mn)

mn.add_cascade(label='File', menu=file_menu)

file_menu.add_command(label='New')

file_menu.add_command(label='Open...')

file_menu.add_command(label='Save')

file_menu.add_separator()

file_menu.add_command(label='About')

file_menu.add_separator()

file_menu.add_command(label='Exit', command=win.quit)

help_menu = Menu(mn)

mn.add_cascade(label='Help', menu=help_menu)

help_menu.add_command(label='Feedback')

help_menu.add_command(label='Contact')

win.mainloop() #running the loop that works as a trigger
```

Output:





8.10 Message:

This widget is similar to a label. Here we can display multiline and non-editable text. Its syntax is:

msg=Message(master, option=value)

The parent window is given as the 'master' parameter. The other parameters that let us set other features are:

A. bd: This sets the border around the message widget

B. bg: We can set the background color of the message using this parameter

C. font: This parameter sets the font style in the message

D. image: This parameter sets the background image of the message widget

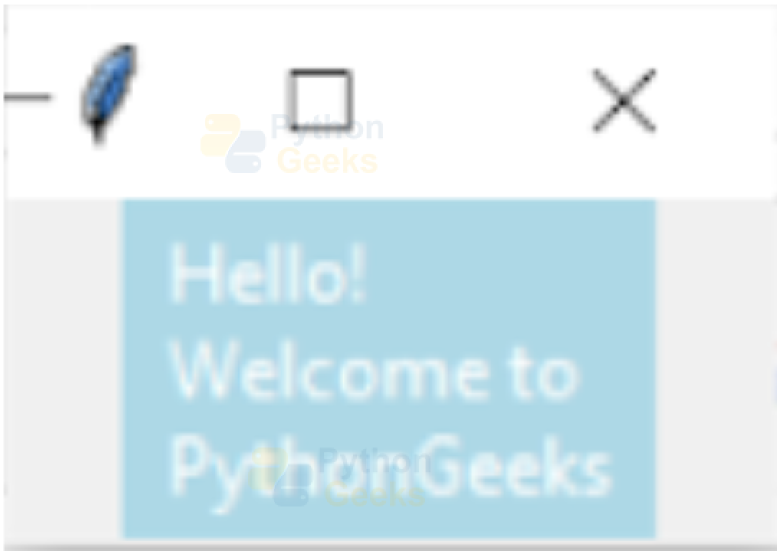
E. width: This sets the width of the message

F. height: This sets the height of the message

Let us see an example.

```
from tkinter import *

win=Tk() #creating the main window and storing the window object in 'win'

message ='Hello! Welcome to PythonGeeks'

msg = Message(win, text = message)

msg.config(bg='lightblue',fg='white')

msg.pack( )

win.mainloop() #running the loop that works as a trigger
```

Output:



```
Hello!
Welcome to
PythonGeeks
```

Radiobutton: This widget lets us give multiple options to the user and he/she can choose one of the options. Its syntax is:

rb=Radiobutton(master, option=value)

The parent window is given as the 'master' parameter. The other parameters that let us set other features are:

A. activebackground: This parameter sets the background color when the cursor is on the radio button

B. activeforeground: This parameter sets the foreground color when the cursor is on the radio button

C. bg: We can set the background color of the radio button using this parameter

D. command: This takes a function that contains the code to set the functionality of the radio button.

E. font: This parameter sets the font style if the items in the radio button

F. image: This parameter sets the background image of the radio button widget

G. width: This sets the width of the radio button

H. height: This sets the height of the radio button

Example of the radio button:

```
from tkinter import *
```

win=Tk() #creating the main window and storing the window object in 'win'

var = IntVar()

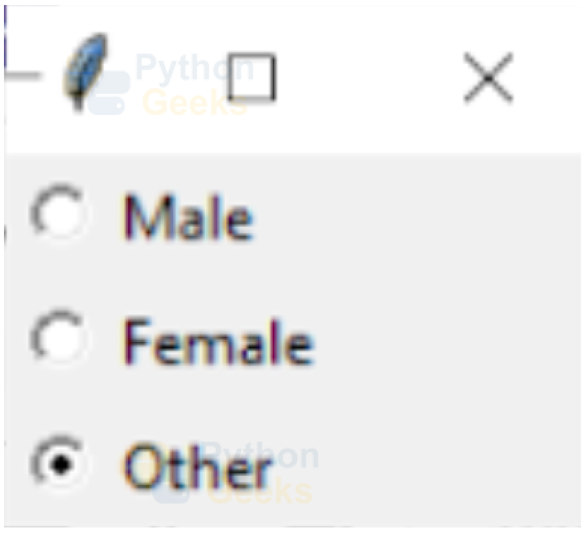
Radiobutton(win, text='Male', variable=var, value=1).pack(anchor=W)

Radiobutton(win, text='Female', variable=var, value=2).pack(anchor=W)

Radiobutton(win, text='Other', variable=var, value=3).pack(anchor=W)

win.mainloop() #running the loop that works as a trigger

Output:



8.11 Scale:

This provides a slider that allows the user to slide any value. Its syntax is:

sc=Scale(master, option=value)

The parent window is given as the 'master' parameter. The other parameters that let us set other features are:

A. activebackground: This parameter sets the background color when the cursor is on the scale

B. bg: We can set the background color of the scale using this parameter

C. cursor: This sets the cursor used in the scale

D. from_: This sets the value of one end of the scale

E. image: This parameter sets the background image of the scale

F. orient: Set the orientation of the widget to either HORIZONTAL or VERTICAL. By default, it is vertical.

G. to: It sets the value of the other end of the scale.

H. width: This sets the width of the scale

Example of the scale:

*from tkinter import **

win=Tk() #creating the main window and storing the window object in 'win' *sc1= Scale(win, from_=0, to=15)*

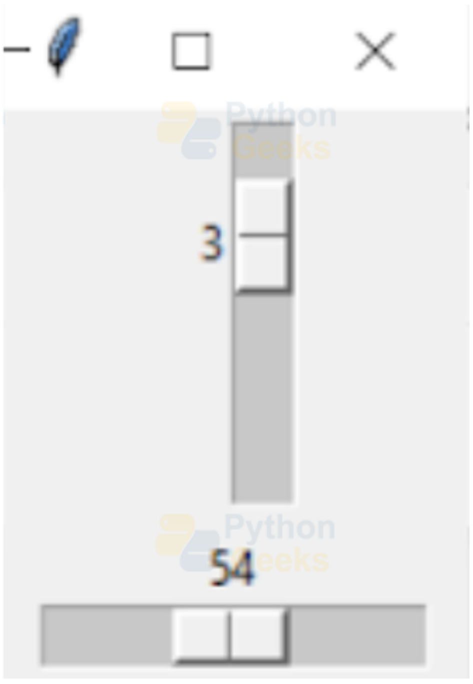
sc1.pack()

sc2 = Scale(win, from_=10, to=100, orient=HORIZONTAL)

sc2.pack()

win.mainloop() #running the loop that works as a trigger

Output:



8.12 Scrollbar:

This widget gives the scrolling feature to the other widgets like lists. Its syntax is:

sb=Scrollbar(master, option=value)

The parent window is given as the 'master' parameter. The other parameters that let us set other features are:

A. activebackground: This parameter sets the background color when the cursor is on the scrollbar

B. bd: This parameter sets the border around the scrollbar

C. bg: We can set the background color of the scrollbar

D. cursor: This sets the cursor used when the cursor is on the scrollbar

E. width: This sets the width of the scrollbar

Example of the scrollbar:

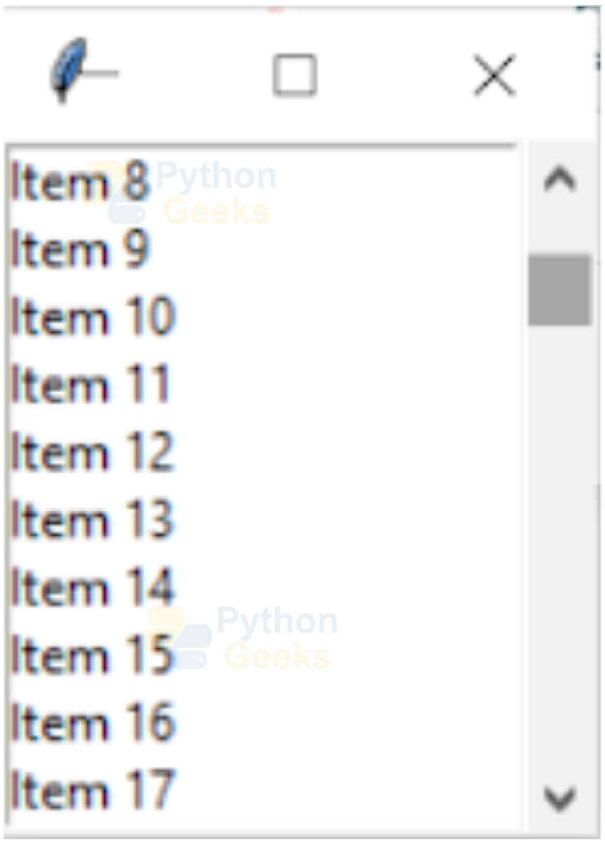
*from tkinter import **

win=Tk() #creating the main window and storing the window object in 'win'

sb = Scrollbar(win)

```
sb.pack( side = RIGHT, fill = Y )

list_1 = Listbox(win, yscrollcommand = sb.set )

for i in range(100):

list_1.insert(END, 'Item ' + str(i))

list_1.pack( side = LEFT, fill = BOTH )

sb.config( command = list_1.yview )

win.mainloop() #running the loop that works as a trigger
```

Output:



```
Item 8
Item 9
Item 10
Item 11
Item 12
Item 13
Item 14
Item 15
Item 16
Item 17
```

8.13 Text:

This is a widget that can be used to display multiline text. Its syntax is:

txt=Text(master, option=value)

The parent window is given as the 'master' parameter. The other parameters are:

A. insertbackground: This parameter sets the background color of text

B. bg: We can set the background color of the test

C. font: This sets the font style of the text in this widget

D. highlightcolor: This sets the highlight color when the cursor is on this widget

E. height: This sets the height of the widget

F. image: This sets the image on the text widget

G. width: This sets the width of the text

Example of the text:

*from tkinter import **

win=Tk() #creating the main window and storing the window object in 'win'

txt= Text(win, height=5, width=45)

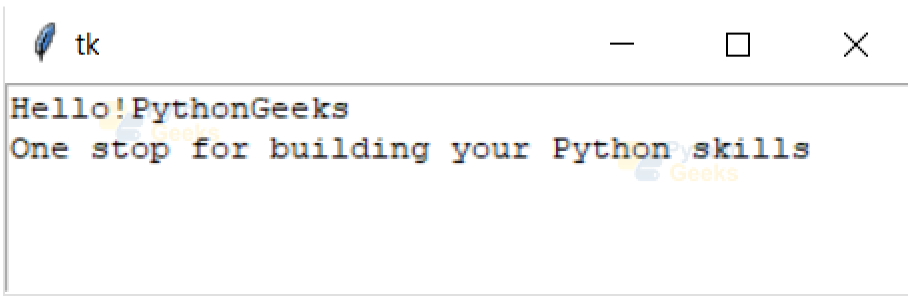
txt.pack()

txt.insert(INSERT,"Hello!")

*txt.insert(**END**, 'PythonGeeks\nOne stop for building your Python skills\n')*

win.mainloop() #running the loop that works as a trigger

Output:



```
Hello!PythonGeeks
One stop for building your Python skills
```

8.14 TopLevel:

This widget provides a separate container window. This does not need a parent window. Its syntax is:

tl=TopLevel(master, option=value)

The parent window is given as the 'master' parameter. The other parameters that let us set other features are:

A. bd: This parameter sets the border around the top level

B. bg: We can set the background color of the top level

C. cursor: This sets the cursor used when the cursor is on the top level

D. height: This sets the height of the widget

E. width: This sets the width of the top level

Example of the top level:

*from tkinter import **

win=Tk() #creating the main window and storing the window object in 'win'

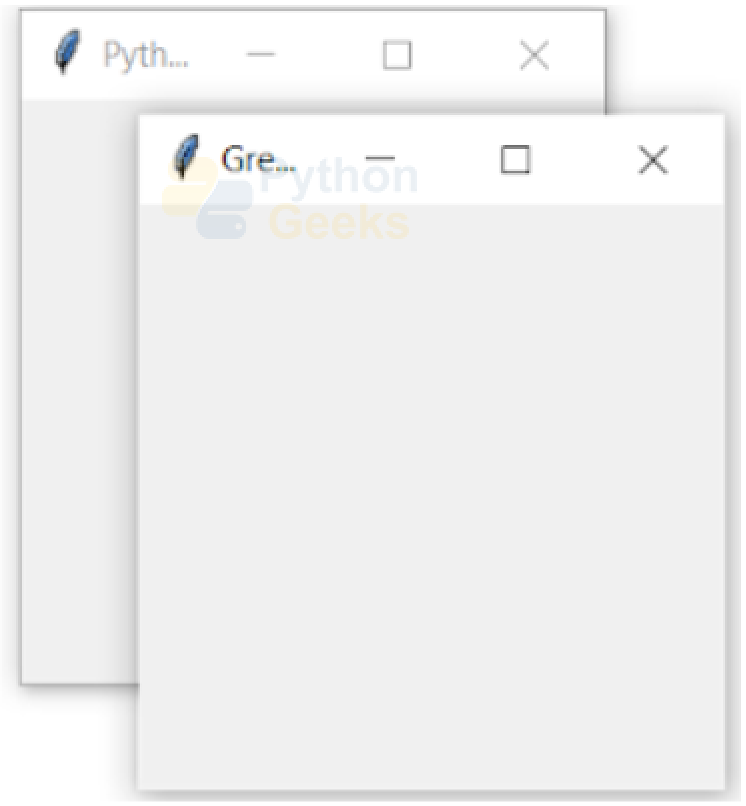
win.title('PythonGeeks')

tl= Toplevel()

tl.title('Greetings')

win.mainloop() #running the loop that works as a trigger

Output:



8.15 SpinBox:

This is similar to the entry widget. Here, instead of entering random input, the user is bound to enter any of the fixed numbers. Its syntax is:

sb=SpinBox(master, option=value)

The parent window is given as the 'master' parameter. The other parameters that let us set other features are:

A. activebackground: This parameter sets the background color when the cursor is on the SpinBox

B. bd: This parameter sets the border around the SpinBox

C. bg: We can set the background color of the SpinBox

D. command: This takes a function that contains the code to set the functionality of the SpinBox

E. cursor: This sets the cursor used when the cursor is on the SpinBox

F. from_: This sets the value of one end of the SpinBox

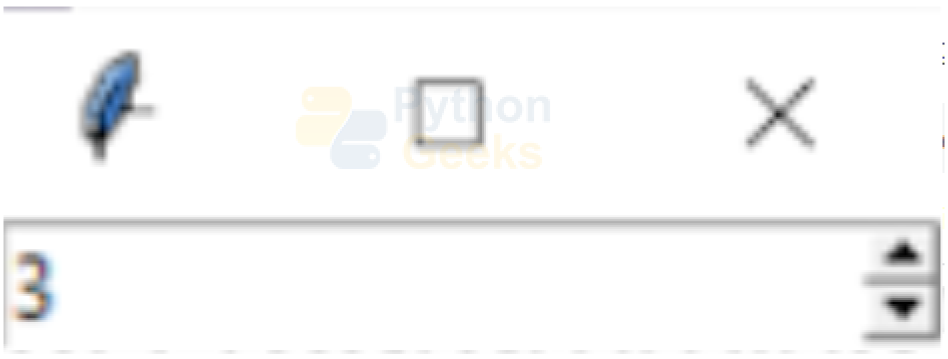
G. to: It sets the value of the other end of the SpinBox.

H. width: This sets the width of the SpinBox

Example of the SpinBox:

```
from tkinter import *

win=Tk() #creating the main window and storing the window object in 'win'

sb = Spinbox(win, from_ = 1, to = 15)

sb.pack()

win.mainloop() #running the loop that works as a trigger
```

Output:



8.16 PannedWindow:

This is a widget that acts as a container of multiple panes oriented horizontally or vertically. Its syntax is:

pw=PannedWindow(master, option=value)

The parent window is given as the 'master' parameter. The other parameters that let us set other features are:

A. bd: This parameter sets the border around the PannedWindow

B. bg: We can set the background color of the PannedWindow

C. cursor: This sets the cursor used when the cursor is on the PannedWindow

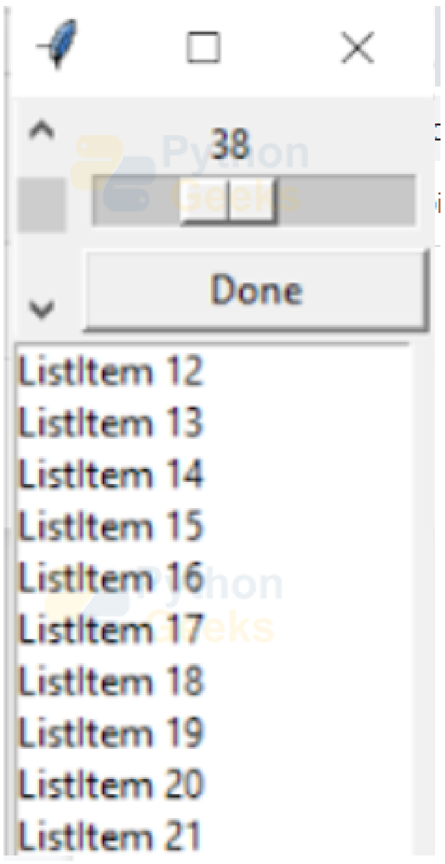
D. width: This sets the width of the PannedWindow

Example of the PannedWindow:

```
from tkinter import *

win=Tk() #creating the main window and storing the window object in 'win'

pw = PanedWindow()

pw.pack(fill = BOTH, expand = 1)

w1 = Scrollbar(win)

w1.pack( side = RIGHT, fill = Y )

list_1 = Listbox(win, yscrollcommand = w1.set )
```

```
for i in range(50):
list_1.insert(END, 'ListItem ' + str(i))
list_1.pack( side = LEFT, fill = BOTH )
w1.config( command = list_1.yview )
pw.add(w1)
w2 = PanedWindow(pw, orient = VERTICAL)
pw.add(w2)
sc = Scale( w2, orient = HORIZONTAL)
w2.add(sc)
w3=Button(w2,text="Done")
w2.add(w3)
win.mainloop() #running the loop that works as a trigger
```

Output:



8.17 LabelFrame:

This widget acts as a container for complex window layouts. Its syntax is:

lf=LabelFrame(master, option=value)

The parent window is given as the ‘master’ parameter. The other parameters that let us set other features are:

A. bd: This parameter sets the border around the LabelFrame

B. bg: We can set the background color of the LabelFrame

C. cursor: This sets the cursor used when the cursor is on the LabelFrame

D. font: This sets the font style of the text in this widget

E. highlightbackground: This sets the highlight color when the widget is not in focus

F. highlightcolor: This sets the highlight color when the cursor is on this widget

G. height: This sets the height of the widget

H. labelAnchor: This is sed to specify where to place the label

I. text: This specifies the text to be displayed in the widget

J. width: This sets the width of the LabelFrame

Example of the LabelFrame:

*from tkinter import **

win=Tk() #creating the main window and storing the window object in 'win'

lf=LabelFrame(win,text="Message")

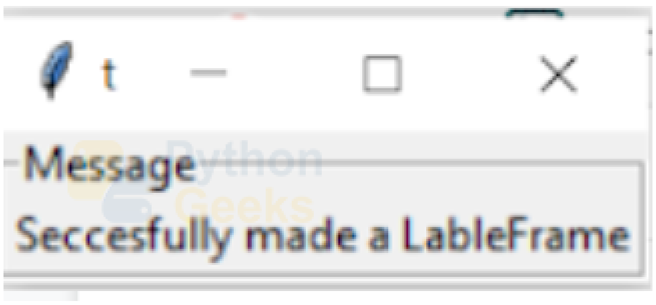
lf.pack(fill="both",expand="yes")

lb=Label(lf,text="Seccesfully made a LableFrame")

lb.pack()

win.mainloop()

Output:



```
Message
Seccesfully made a LableFrame
```

tkMessageBox: This widget is used to show pop-up messages. Remember we used this widget while discussing the button widget! The syntax of this widget is:

mb=tkMessageBox.FunctionName(title, message [, options])

In this, the function name can be any of the following functions:

A. askquestion()

B. askokcancel()

C. askyesno ()

D. askretrycancel ()

E. showerror ()

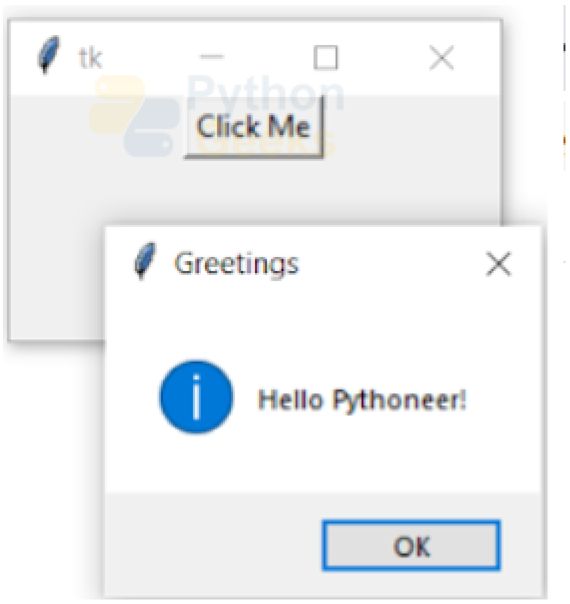
F. showinfo()

G. showwarning()

Example of messagebox:

```
from tkinter import *

from tkinter import messagebox

win=Tk() #creating the main window and storing the window object in 'win'

win.geometry("200x100")

def greet():

messagebox.showinfo("Greetings", "Hello Pythoneer!")

btn = Button(win, text = "Click Me", command = greet)

btn.pack()

win.mainloop()
```

Output:



Hello Pythoneer!