# Tkinter & SqLite3

# ABOUT

Tk/Tcl has long been an integral part of Python. It provides a robust and platform independent windowing toolkit, that is available to Python programmers using the tkinter package, and its extension, the tkinter.tix and the tkinter.ttk modules.

tkinter chief virtues are that it is fast, and that it usually comes bundled with Python.

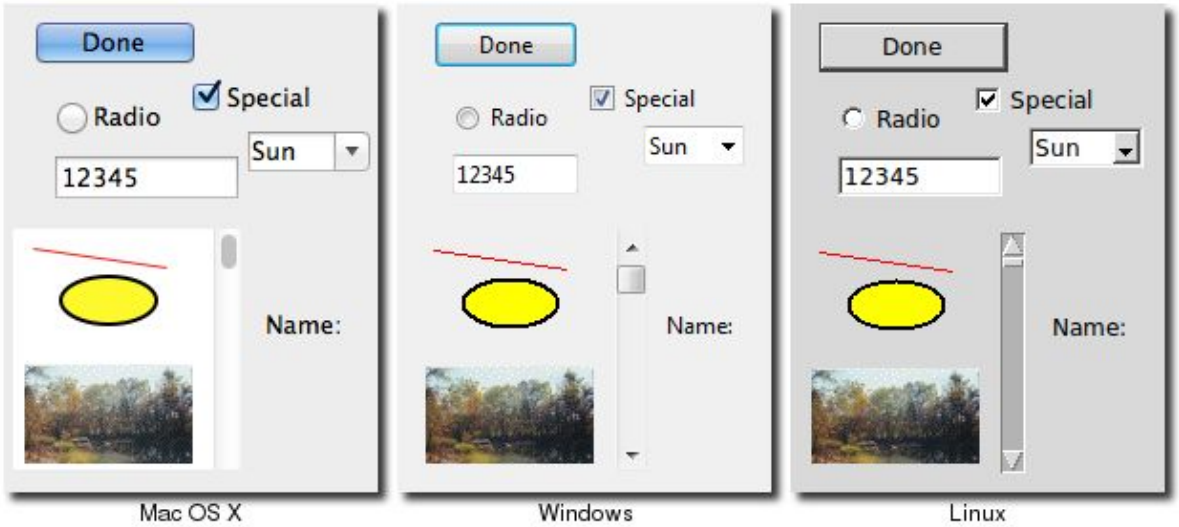
# Creating GUI application using Tkinter is easy task.

- Import the Tkinter module.
- Create the GUI application main window.
- Add one or more widgets to the GUI application.
- Enter the main event loop to take action against each event triggered by the user.

```
import tkinter # in Python 3
top = tkinter.Tk()
# Code to add widgets will go here...
top.mainloop()
```

# Widgets

Widgets are all the things that you see onscreen. In our example, we had a button, an entry, a few labels, and a frame. Others are things like checkboxes, tree views, scrollbars, text areas, and so on.



# Widgets

- button
- canvas
- checkbutton
- combobox
- entry
- frame
- label
- labelframe
- listbox
- menu
- menubutton
- message
- notebook
- tk_optionMenu
- panedwindow
- progressbar
- radiobutton
- scale
- scrollbar
- separator
- sizegrip
- spinbox
- text
- treeview

# Variable Classes

- Some widgets (like text entry widgets, radio buttons and so on) can be connected directly to application variables by using special options: variable, textvariable, onvalue, offvalue, and value.
- This connection works both ways: if the variable changes for any reason, the widget it's connected to will be updated to reflect the new value.
- It's not possible to hand over a regular Python variable to a widget through a variable or textvariable option. The only kinds of variables for which this works are variables that are subclassed from a class called Variable, defined in the Tkinter module.

# Variable Classes

x = StringVar() # Holds a string; default value ""

x = IntVar() # Holds an integer; default value 0

x = DoubleVar() # Holds a float; default value 0.0

x = BooleanVar() # Holds a boolean, returns 0 for False and 1 for True

To read the current value of such a variable, call the method get(). The value of such a variable can be changed with the set() method.

# Geometry strings

A geometry string has this general form: 'wxh±x±y'

The w and h parts give the window width and height in pixels. They are separated by the character 'x'.

If the next part has the form +x, it specifies that the left side of the window should be x pixels from the left side of the desktop. If it has the form -x, the right side of the window is x pixels from the right side of the desktop.

If the next part has the form +y, it specifies that the top of the window should be y pixels below the top of the desktop. If it has the form -y, the bottom of the window will be y pixels above the bottom edge of the desktop.

# Geometry strings

For example, a window created with geometry='120x50-0+20' would be 120 pixels wide by 50 pixels high, and its top right corner will be along the right edge of the desktop and 20 pixels below the top edge.

```
import tkinter

top = tkinter.Tk()

top.geometry('120x50-0+20')

top.mainloop()
```

```
import tkinter
top = tkinter.Tk()
#to have our gui fullscreen of a window
screen_width = top.winfo_screenwidth()
screen_height = top.winfo_screenheight()
screen_resolution =
str(screen_width)+'x'+str(screen_height)
top.geometry(screen_resolution)
top.mainloop()
```

# Adding title to root window

```
import tkinter

top = tkinter.Tk()

top.title("Home")

top.mainloop()
```

# Button widget

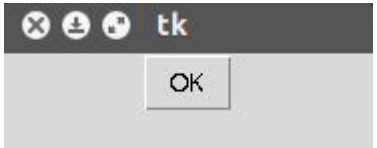
The Button widget is used to add buttons in a Python application. These buttons can display text or images that convey the purpose of the buttons. You can attach a function or a method to a button which is called automatically when you click the button.

```
import tkinter
parent = tkinter.Tk()
B = tkinter.Button(parent, text ="Hello")
B.pack()
parent.mainloop()
```

# Button widget

**Tk button with onClick event :** To create a Tkinter window with a button use the example below. The program enters mainloop() which wait for events (user actions). We define the button which has a callback to the function callback(). master is the root window, the window where your button will appear in.

```
from tkinter import *
master = Tk()
master.geometry('120x50-0+20')
def callback():
    print ("click!")
b = Button(master, text="OK",
command=callback)
b.pack()
mainloop()
```



# Label Widget

This widget implements a display box where you can place text or images. The text displayed by this widget can be updated at any time you want.

It is also possible to underline part of the text (like to identify a keyboard shortcut) and span the text across multiple lines.

w = Label ( master, option, ... )

```
import tkinter
parent = tkinter.Tk()
l = tkinter.Label(parent, text="This is Label Widget")
l.pack()
parent.mainloop()
```

# Label Widget

Colorized Labels in various fonts

Some Tk widgets, like the label, text, and canvas widget, allow you to specify the fonts used to display text. This can be achieved by setting the attribute "font". typically via a "font" configuration option. You have to consider that fonts are one of several areas that are not platform-independent.

The attribute fg can be used to have the text in another colour and the attribute bg can be used to change the background colour of the label.

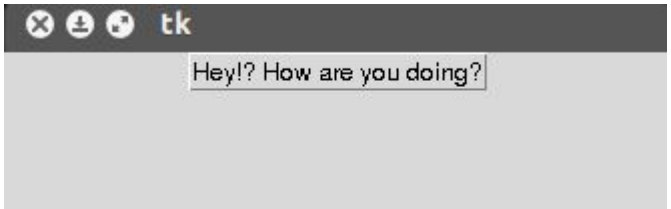
# Label Widget

```
import tkinter as tk

root = tk.Tk()

tk.Label(root, text="Red Text in Times Font",fg = "red",font = "Times").pack()

tk.Label(root, text="Green Text in Helvetica Font",fg = "light green",bg = "dark green",font = "Helvetica 16 bold italic").pack()

tk.Label(root,text="Blue Text in Verdana bold",fg = "blue", bg = "yellow",font = "Verdana 10 bold").pack()

root.mainloop()
```

# Label Widget

You can associate a Tkinter variable with a label. When the contents of the variable changes, the label is automatically updated:

```
from tkinter import *

root = Tk()

var = StringVar()

label = Label( root, textvariable=var, relief=RAISED )

var.set("Hey!? How are you doing?")

label.pack()

root.mainloop()
```



# Entry widget

The Entry widget is used to accept single-line text strings from a user.

If you want to display multiple lines of text that can be edited, then you should use the Text widget.

Syntax

Here is the simple syntax to create this widget −

w = Entry( master, option, ... )

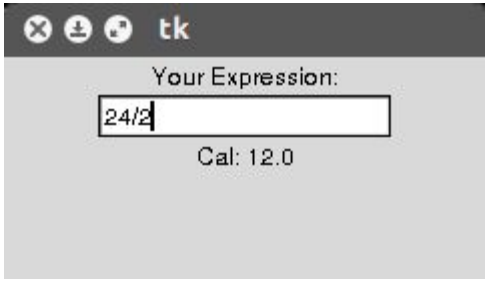
# Entry widget

```
import tkinter

parent = tkinter.Tk()

e = tkinter.Entry(parent)

e.pack()

parent.mainloop()
```

```
from tkinter import *

def show_entry_fields():

   print("First Name: %s\nLast Name: %s" % (e1.get(), e2.get()))

master = Tk()

l1= Label(master, text="First Name")

l2 = Label(master, text="Last Name")

e1 = Entry(master)

e2 = Entry(master)

l1.pack()

l2.pack()

e1.pack()

e2.pack()
```


```
b= Button(master, text='Show', command=show_entry_fields)
b.pack()
mainloop( )
```

```
from tkinter import *

from math import *

def evaluate(event):

    res.configure(text = "Cal: " + str(eval(entry.get())))

w = Tk()

Label(w, text="Your Expression:").pack()

entry = Entry(w)

entry.bind("<Return>", evaluate)

entry.pack()

res = Label(w)

res.pack()

w.mainloop()
```



# Message widget

This widget provides a multi line and non editable object that displays texts, automatically breaking lines and justifying their contents.

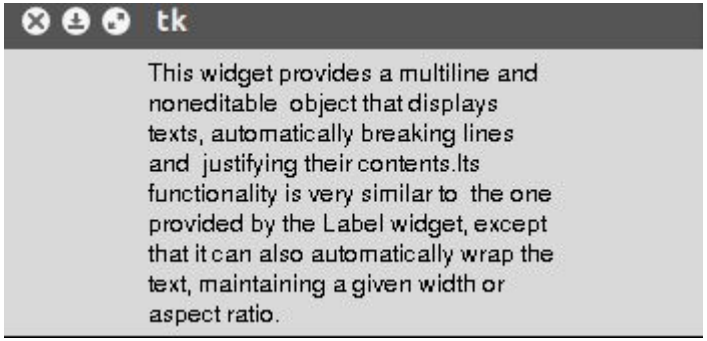
Its functionality is very similar to the one provided by the Label widget, except that it can also automatically wrap the text, maintaining a given width or aspect ratio.

Syntax

Here is the simple syntax to create this widget −

w = Message ( master, option, ... )

```
import tkinter

parent = tkinter.Tk()

var = tkinter.StringVar()

m = tkinter.Message(parent, textvariable=var)

var.set("This widget provides a multiline and noneditable \
 object that displays texts, automatically breaking lines and \
 justifying their contents.Its functionality is very similar to \
 the one provided by the Label widget, except that it can also\
 automatically wrap the text, maintaining a given width or aspect ratio.")

m.pack()

parent.mainloop()
```



# Layout Managers / Geometry Manager

Tkinter possess three layout managers:

pack

grid

place

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

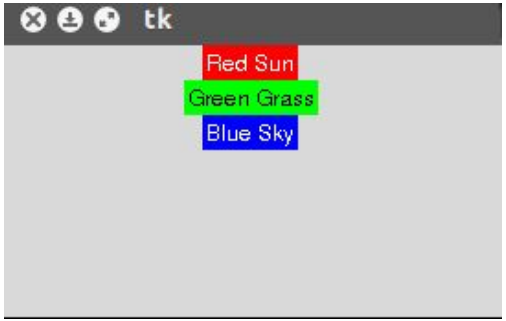
The place geometry manager positions widgets using absolute positioning. The pack geometry manager organises widgets in horizontal and vertical boxes. The grid geometry manager places widgets in a two dimensional grid.

# 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.

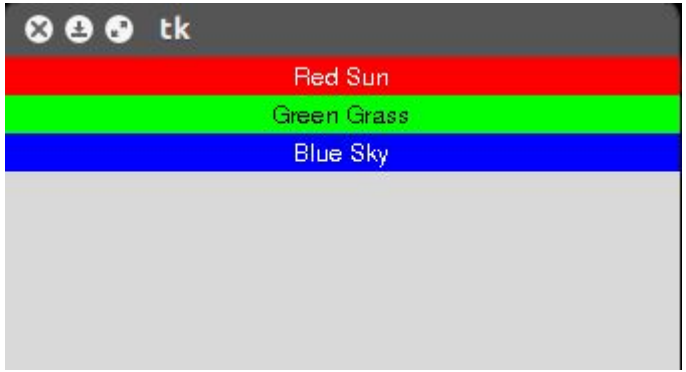
For simple applications it is definitely the manager of choice. For example simple applications like placing a number of widgets side by side, or on top of each other.

```
from tkinter import *

root = Tk()

w = Label(root, text="Red Sun", bg="red", fg="white")

w.pack()

w = Label(root, text="Green Grass", bg="green", fg="black")

w.pack()

w = Label(root, text="Blue Sky", bg="blue", fg="white")

w.pack()

mainloop()
```



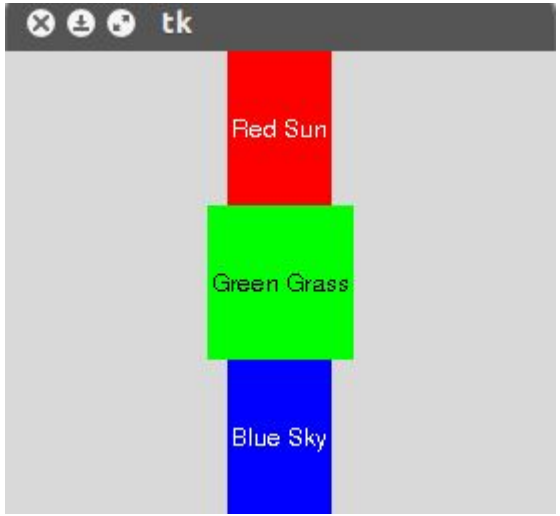
each label has been given the size of the text. If you want to make the widgets as wide as the parent widget, you have to use the fill=X option:

```
from tkinter import *
root = Tk()
w = Label(root, text="Red Sun", bg="red",
fg="white")
w.pack(fill=X)
w = Label(root, text="Green Grass", bg="green",
fg="black")
w.pack(fill=X)
w = Label(root, text="Blue Sky", bg="blue",
fg="white")
w.pack(fill=X)
root.mainloop()
```



If you want to make the widgets as long as the parent widget, you have to use the fill=Y option and expand=True , so when you resize your screen your widgets expands

```
from tkinter import *
root = Tk()
w = Label(root, text="Red Sun", bg="red",
fg="white")
w.pack(fill=Y,expand=True)
w = Label(root, text="Green Grass", bg="green",
fg="black")
w.pack(fill=Y,expand=True)
w = Label(root, text="Blue Sky", bg="blue",
fg="white")
w.pack(fill=Y,expand=True)
root.mainloop()
```



If you want to make the widgets as long and as wide as the parent widget, you have to use the fill=BOTH option and expand=True , so when you resize your screen your widgets expands

```
from tkinter import *
root = Tk()
w = Label(root, text="Red Sun", bg="red",
fg="white")
w.pack(fill=BOTH,expand=True)
w = Label(root, text="Green Grass", bg="green",
fg="black")
w.pack(fill=BOTH,expand=True)
w = Label(root, text="Blue Sky", bg="blue",
fg="white")
w.pack(fill=BOTH,expand=True)
root.mainloop()
```

# SQLite3

Importing sqlite

```
import sqlite3
```

Connecting with Sqlite3:

```
with sqlite3.connect("details.db") as db:
    cursor = db.cursor()
```

Executing Sql statements

```
cursor.execute(" create table if not exists users(username text primary key); ")
```

Store result in variable

```
cursor.execute("select count(*) from users where username='"+user+"'")
result= cursor.fetchone()
```

# Experiment 12:

- Create a login page in tkinter
- Connect it with sqlite3 to accept new users and verify existing users