

Graphical User Interface I

#12



Serdar ARITAN

Biomechanics Research Group,
Faculty of Sports Sciences, and
Department of Computer Graphics
Hacettepe University, Ankara, Turkey

GUI : Using Module Tkinter

Python's standard library includes Tcl/Tk—Tcl is an almost syntax-free scripting language and **Tk** is a **GUI library written in Tcl and C**. Python's tkinter module provides Python bindings for the **Tk** GUI library. Tk has three advantages compared with the other GUI libraries that are available for Python.

First, **it is installed as standard with Python**, so it is always available;
second, **it is small** (even including Tcl);
and **third**, it comes with IDLE which is very useful for experimenting with Python and for editing and debugging Python programs.



HOME	ABOUT TCL/TK	SOFTWARE	CORE DEVELOPMENT	COMMUNITY	DOCUMENTATION
------	--------------	----------	------------------	-----------	---------------

SEARCH

GO

Welcome to the Tcl Developer Xchange!

Join the many thousands of software developers who are already more productive with help from the **Tcl programming language** and the **Tk graphical user interface toolkit**.

Tcl (Tool Command Language) is a very powerful but easy to learn dynamic programming language, suitable for a very wide range of uses, including web and desktop applications, networking, administration, testing and many more. Open source and business-friendly, Tcl is a mature yet evolving language that is truly cross platform, easily deployed and highly extensible.

Tk is a graphical user interface toolkit that takes developing desktop applications to a higher level than conventional approaches. Tk is the standard GUI not only for Tcl, but for many other dynamic languages, and can produce rich, native applications that run unchanged across Windows, Mac OS X, Linux and more.

- [Learn more](#)
- [Get Tcl/Tk \(9.0\) \(8.6\)](#)
- Browse the [Tcler's Wiki](#)
- Read the [reference pages](#) and [other documentation](#)



Tcl Conference News

The 20th European OpenACS and Tcl conference took place in Vienna, on July 11th and 12th 2024

Please visit the [main conference page](#) for details.

[Older conference info](#)

Latest Software Releases

[Tcl/Tk 9.0.0](#) *Source*

[Tcl/Tk 8.6.15](#) *Source*

[All Tcl/Tk Downloads](#)

[ActiveTcl](#) *Multi-platform and commercially supported*

[BAWT](#) *Multi-platform*

[Magicspat](#) *Windows*

[IronTcl](#) *Windows*

[Tklib 0.8](#) *Jul 6, 2024*

[Tcllib 2.0](#) *Sep 26, 2024*

[Wiki Category: Distributions](#)

GUI : Using Module Tkinter

The three most well-established crossplatform GUI libraries with Python bindings are PyGtk (www.pygtk.org), PyQt (www.riverbankcomputing.com/software/pyqt), and wxPython (www.wxpython.org). All three of these offer far more widgets than Tk, produce better-looking GUIs and make it possible to create custom widgets drawn in code.

All of them are easier to learn and use than Tk and all have more and much better Python-oriented documentation than Tk. And in general, programs that use **PyGtk**, **PyQt**, or **wxPython** need less code and produce better results than programs written using **Tk**.

Every **tkinter** program consists of these things:

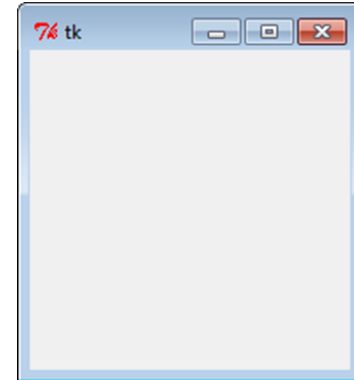
- **Windows, buttons, scrollbars, text areas, and other** widgets—anything that you can see on the computer screen (Generally, the term widget means any useful object; in programming, it is short for “window gadget.”)
- **Modules, functions**, and classes that manage the data that is being shown in the GUI—you are familiar with these; they are the tools you’ve seen so far in this book.
- An **event manager** that listens for events such as mouse clicks and keystrokes and reacts to these events by calling event handler functions

Tk was written by John Ousterhout while at Berkeley.

Tkinter was written by **Steen Lumholt** and **Guido van Rossum**.

Here is a small but complete tkinter program:

```
import tkinter
window = tkinter.Tk()
# Code to add widgets will go here...
window.mainloop()
```



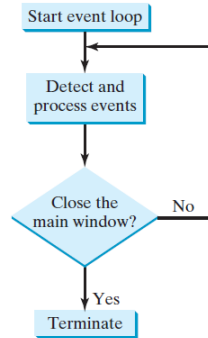
Tk is a class that represents the root window of a tkinter GUI. This root window's mainloop method handles all the events for the GUI, so it's important to create **only one instance of Tk**.

The root window is initially empty. If the window on the screen is closed, the window object is destroyed (though we can create a new root window by calling Tk() again).

GUI : Using Module Tkinter

The call on method `mainloop` doesn't exit until the window is destroyed (which happens when you click the appropriate widget in the title bar of the window), so any code following that call won't be executed until later:

```
import tkinter
window = tkinter.Tk()
window.mainloop()
print('Hello')
```



When you try this code, you'll see that the call on function `print` doesn't get executed until after the window is destroyed. That means that if you want to make changes to the GUI after you have called `mainloop`, you need to do it in an event-handling function.

Widget	Description
Button	A clickable button
Canvas	An area used for drawing or displaying images
Checkbutton	A clickable box that can be selected or unselected
Entry	A single-line text field that the user can type in
Frame	A container for widgets
Label	A single-line display for text
Listbox	A drop-down list that the user can select from
Menu	A drop-down menu
Message	A multiline display for text
Menubutton	An item in a drop-down menu
Text	A multiline text field that the user can type in
TopLevel	An additional window

window : This term has different meanings in different contexts, but in general it refers to a rectangular area somewhere on your display screen.

top-level window : A window that exists independently on your screen. It will be decorated with the standard frame and controls for your system's desktop manager.

widget : The **generic term** for any of the building blocks that make up an application in a graphical user interface.

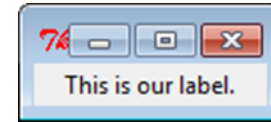
frame : In tkinter, the Frame widget is the basic unit of organization for complex layouts.

child, parent : When any widget is created, a parent-child relationship is created. For example, if you place a text label inside a frame, the frame is the parent of the label.



Labels are widgets that are used to display short pieces of text.

```
import tkinter
window = tkinter.Tk()
label = tkinter.Label(window, text='This is our label.')
label.pack()
window.mainloop()
```



Method call `label.pack()` is crucial. Each widget has a method called pack that places it in its parent widget and then tells the parent to resize itself as necessary. If we forget to call this method, the child widget (in this case, Label) won't be displayed or will be displayed improperly.



Labels display text. Often, applications will want to update a label's text as the program runs to show things like the name of a file or the time of day. One way to do this is simply to assign a new value to the widget's text using method `config`:

```
>>> import tkinter
>>> window = tkinter.Tk()
>>> label = tkinter.Label(window, text='First Label.')
>>> label.pack()
>>> label.config(text='Second Label.')
```



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

Option	Description
anchor	This options controls where the text is positioned if the widget has more space than the text needs. The default is anchor=CENTER, which centers the text in the available space.
bg	The normal background color displayed behind the label and indicator.
bitmap	Set this option equal to a bitmap or image object and the label will display that graphic.
bd	The size of the border around the indicator. Default is 2 pixels.
cursor	If you set this option to a cursor name (<i>arrow</i> , <i>dot</i> etc.), the mouse cursor will change to that pattern when it is over the checkbutton.
font	If you are displaying text in this label (with the text or textvariable option, the font option specifies in what font that text will be displayed.
fg	If you are displaying text or a bitmap in this label, this option specifies the color of the text. If you are displaying a bitmap, this is the color that will appear at the position of the 1-bits in the bitmap.
height	The vertical dimension of the new frame.
image	To display a static image in the label widget, set this option to an image object.
justify	Specifies how multiple lines of text will be aligned with respect to each other: LEFT for flush left, CENTER for centered (the default), or RIGHT for right-justified.
padx	Extra space added to the left and right of the text within the widget. Default is 1.
pady	Extra space added above and below the text within the widget. Default is 1.
relief	Specifies the appearance of a decorative border around the label. The default is FLAT; for other values.
text	To display one or more lines of text in a label widget, set this option to a string containing the text. Internal newlines ("n") will force a line break.
textvariable	To save the text displayed in a label widget to a control variable of class <i>StringVar</i> , set this option to that variable.
underline	You can display an underline () below the nth letter of the text, counting from 0, by setting this option to n. The default is underline=-1, which means no underlining.
width	Width of the label in characters (not pixels!). If this option is not set, the label will be sized to fit its contents.
wraplength	You can limit the number of characters in each line by setting this option to the desired number. The default value, 0, means that lines will be broken only at newlines.



```
from tkinter import *  
window = Tk()
```

```
label = Label(window, text = 'Hello',  
               background = 'white',  
               foreground = 'red',  
               font = 'Times 20',  
               relief = 'groove',  
               borderwidth = 3)
```

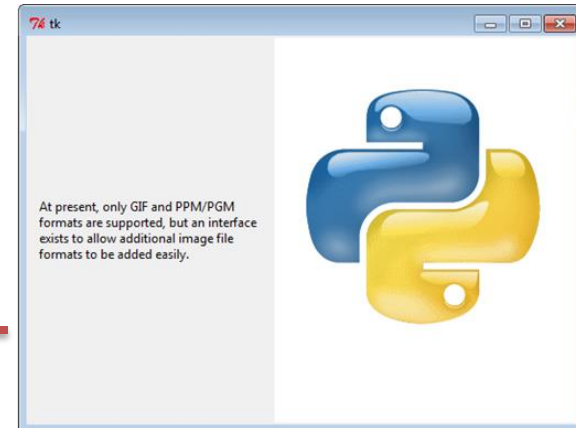
```
label.grid(row=0, column=0)  
mainloop()
```



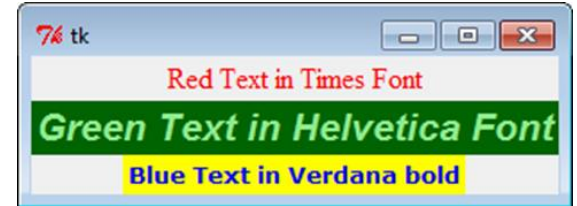
```
from tkinter import *
```

```
root = Tk()
logo = PhotoImage(file="python-logo-glassy.gif")
w1 = Label(root, image=logo).pack(side="right")
explanation = """At present, only GIF and PPM/PGM
    formats are supported, but an interface
    exists to allow additional image file
    formats to be added easily."""
w2 = Label(root,
            justify=LEFT,
            padx = 10,
            text=explanation).pack(side="left")

root.mainloop()
```



Try using grid



```
from tkinter import *
```

```
root = Tk()
```

```
Label(root,
```

```
    text="Red Text in Times Font",
```

```
    fg = "red",
```

```
    font = "Times").pack()
```

```
Label(root,
```

```
    text="Green Text in Helvetica Font",
```

```
    fg = "light green",
```

```
    bg = "dark green",
```

```
    font = "Helvetica 16 bold italic").pack()
```

```
Label(root,
```

```
    text="Blue Text in Verdana bold",
```

```
    fg = "blue",
```

```
    bg = "yellow",
```

```
    font = "Verdana 10 bold").pack()
```

```
root.mainloop()
```



```
import tkinter as tk
```

```
counter = 0
```

```
def counter_label(label):
```

```
    def count():
```

```
        global counter
```

```
        counter += 1
```

```
        label.config(text=str(counter))
```

```
        label.after(1000, count)
```

```
    count()
```

```
root = tk.Tk()
```

```
root.title("Counting Seconds")
```

```
label = tk.Label(root, fg="blue")
```

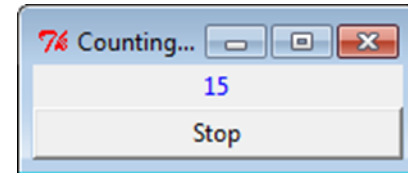
```
label.pack()
```

```
counter_label(label)
```

```
button = tk.Button(root, text='Stop', width=25, command = root.destroy)
```

```
button.pack()
```

```
root.mainloop()
```



Suppose we want to display a string, such as the current time or a score in a game, in several places in a GUI—the application’s status bar, some dialog boxes, and so on. Calling method `config` on each widget every time there is new information isn’t hard, but as the application grows, so too do the odds that we’ll forget to update at least one of the widgets that’s displaying the string. What we really want is a string that “knows” which widgets care about its value and can alert them itself when that value changes.

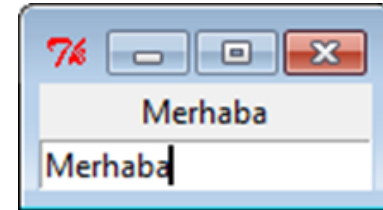
Python’s **strings**, **integers**, **floating-point numbers**, and **Booleans** are **immutable**, so module `tkinter` provides one class for each of the **immutable** types: `StringVar` for `str`, `IntVar` for `int`, `BooleanVar` for `bool`, and `DoubleVar` for `float`.

```
import tkinter
window = tkinter.Tk()
data = tkinter.StringVar()
data.set('Data to display')
label = tkinter.Label(window, textvariable = data)
label.pack()
window.mainloop()
```

Notice that this time we assign to the `textvariable` parameter of the label rather than the `text` parameter. The values in tkinter containers are set and retrieved using the methods `set` and `get`. Whenever a `set` method is called, it tells the label, and any other widgets it has been assigned to, that it's time to update the GUI. There is one small trap here for newcomers: because of the way module `tkinter` is structured, you cannot create a `StringVar` or any other mutable variable until you have created the root Tk window.

Getting Information from the User with the **Entry Type**: Two widgets let users enter text. The simplest one is Entry, which allows for a single line of text. If we associate a `stringVar` with the Entry, then whenever a user types anything into that Entry, the `stringVar`'s value will automatically be updated to the contents of the Entry.

```
import tkinter
window = tkinter.Tk()
frame = tkinter.Frame(window)
frame.pack()
var = tkinter.StringVar()
label = tkinter.Label(frame, textvariable=var)
label.pack()
entry = tkinter.Entry(frame, textvariable=var)
entry.pack()
window.mainloop()
```



Grouping Widgets with the Frame Type: A tkinter Frame is a container, much like the root window is a container. Frames are not directly visible on the screen; instead, they are used to organize other widgets.

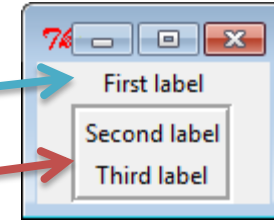
```
import tkinter
window = tkinter.Tk()
frame = tkinter.Frame(window)
frame.pack()
first = tkinter.Label(frame, text='First label')
first.pack()
second = tkinter.Label(frame, text='Second label')
second.pack()
third = tkinter.Label(frame, text='Third label')
third.pack()
window.mainloop()
```



Note that we call pack on every widget; if we omit one of these calls, that widget will not be displayed.



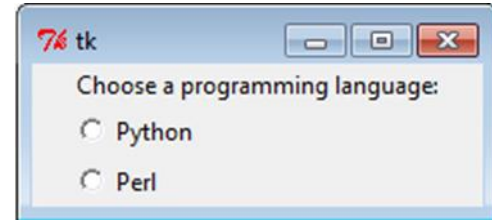
```
import tkinter
window = tkinter.Tk()
frame = tkinter.Frame(window)
frame.pack()
frame2=tkinter.Frame(window, borderwidth=4, relief=tkinter.GROOVE)
frame2.pack()
first = tkinter.Label(frame, text='First label')
first.pack()
second = tkinter.Label(frame2, text='Second label')
second.pack()
third = tkinter.Label(frame2, text='Third label')
third.pack()
window.mainloop()
```



Radio Buttons: sometimes called option button, is a graphical user interface element of Tkinter, which allows the user to choose (exactly) one of a predefined set of options. Radio buttons can contain text or images.

```
from tkinter import *  
root = Tk()  
v = IntVar()
```

```
Label(root,  
      text="Choose a programming language:",  
      justify = LEFT, padx = 20).pack()  
Radiobutton(root, text="Python", padx = 20,  
            variable = v, value=1).pack(anchor=W)  
Radiobutton(root, text="Perl", padx = 20,  
            variable = v, value=2).pack(anchor=W)  
mainloop()
```




```
from tkinter import *  
root = Tk()  
v = IntVar()  
v.set(1)  # initializing the choice, i.e. Python  
languages = [("Python",1), ("Perl",2),  
              ("Java",3), ("C++",4), ("C",5)]  
  
def ShowChoice():  
    print(v.get())
```

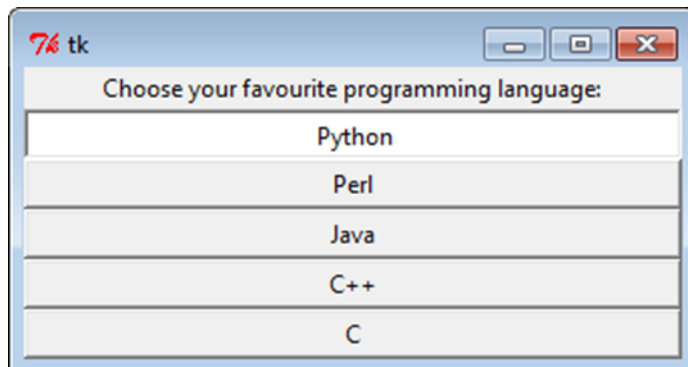
```
Label(root, text="Choose your favourite programming language:",  
       justify = LEFT, padx = 20).pack()
```

```
for txt, val in languages:  
    Radiobutton(root, text=txt, padx = 20, variable=v,  
                command=ShowChoice, value = val).pack(anchor = W)  
mainloop()
```



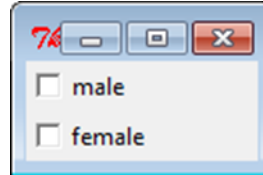


```
for txt, val in languages:  
    Radiobutton(root,  
                text = txt,  
                indicatoron = 0,  
                padx = 20,  
                width = 40,  
                variable = v,  
                command = ShowChoice,  
                value = val).pack(anchor = W)
```



Checkboxes, also known as tickboxes or tick boxes or check boxes, are widgets that permit the user to make multiple selections from a number of different options. This is different to a radio button, where the user can make only one choice.

```
from tkinter import *  
master = Tk()  
var1 = IntVar()  
Checkbutton(master, text="male",  
variable=var1).grid(row=0, sticky=W)  
var2 = IntVar()  
Checkbutton(master, text="female",  
variable=var2).grid(row=1, sticky=W)  
mainloop()
```

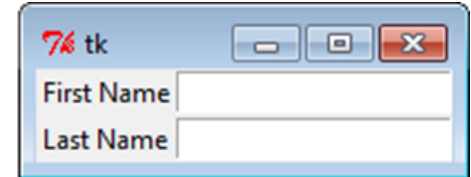


Although there are three different “geometry managers” in tkinter, prefers the `.grid()` geometry manager. This manager treats every window or frame as a table—a gridwork of rows and columns.

Entry widgets are the basic widgets of tkinter used to get input, i.e. text strings, from the user of an application. This widget allows the user to enter a single line of text. If the user enters a string, which is longer than the available display space of the widget, the content will be scrolled. This means, that the string cannot be seen in its entirety.

```
from tkinter import *
```

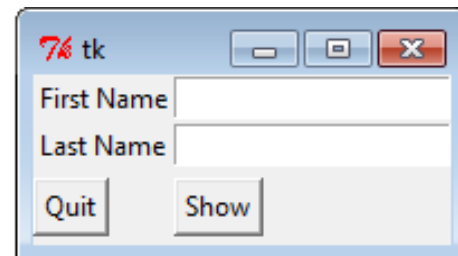
```
master = Tk()  
Label(master, text="First Name").grid(row=0)  
Label(master, text="Last Name").grid(row=1)  
e1 = Entry(master)  
e2 = Entry(master)  
e1.grid(row=0, column=1)  
e2.grid(row=1, column=1)  
mainloop( )
```



```
from tkinter import *
```

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

```
master = Tk()  
Label(master, text="First Name").grid(row=0)  
Label(master, text="Last Name").grid(row=1)  
e1 = Entry(master)  
e2 = Entry(master)  
e1.grid(row=0, column=1)  
e2.grid(row=1, column=1)  
Button(master, text='Quit', command=master.quit).grid(row=3,  
column=0, sticky=W, pady=4)  
Button(master, text='Show', command=show_entry_fields).grid(row=3,  
column=1, sticky=W, pady=4)  
mainloop( )
```



```
from tkinter import *
```

```
def show_entry_fields():  
    print("First Name: %s\nLast Name: %s" % (e1.get(), e2.get()))  
    e1.delete(0,END)  
    e2.delete(0,END)
```

```
master = Tk()  
Label(master, text="First Name").grid(row=0)  
Label(master, text="Last Name").grid(row=1)
```

```
e1 = Entry(master)  
e2 = Entry(master)  
e1.insert(10,"Serdar")  
e2.insert(10,"Aritan")  
e1.grid(row=0, column=1)  
e2.grid(row=1, column=1)
```

```
Button(master, text='Quit', command=master.quit).grid(row=3, column=0, sticky=W, pady=4)  
Button(master, text='Show', command=show_entry_fields).grid(row=3, column=1, sticky=W,  
pady=4)
```

```
mainloop( )
```

The `delete()` method has the format `delete(first, last=None)`. If only one number is given, it deletes the character at index. If two are given, the range from "first" to "last" will be deleted. Use `delete(0, END)` to delete all text in the widget.

we want to start the Entry fields with default values, e.g. we fill in "Serdar" and "Aritan"

The tkinter.ttk module provides access to the Tk themed widget set, introduced in Tk 8.5. To start using Ttk, import its module:

```
from tkinter import ttk
```

To override the basic Tk widgets, the import should follow the Tk import:

```
from tkinter import *  
from tkinter.ttk import *
```

That code causes several tkinter.ttk widgets (Button, Checkbutton, Entry, Frame, Label, LabelFrame, Menubutton, PanedWindow, Radiobutton, Scale and Scrollbar) to automatically replace the Tk widgets.

Ttk comes with 17 widgets, eleven of which already existed in tkinter: Button, Checkbutton, Entry, Frame, Label, LabelFrame, Menubutton, PanedWindow, Radiobutton, Scale and Scrollbar. The other six are new: Combobox, Notebook, Progressbar, Separator, Sizegrip and Treeview. And all them are subclasses of Widget.

tk code:

```
l1 = tkinter.Label(text="Test", fg="black", bg="white")  
l2 = tkinter.Label(text="Test", fg="black", bg="white")
```

ttk code:

```
style = ttk.Style()  
style.configure("BW.TLabel", foreground="black",  
background="white")
```

```
l1 = ttk.Label(text="Test", style="BW.TLabel")  
l2 = ttk.Label(text="Test", style="BW.TLabel")
```

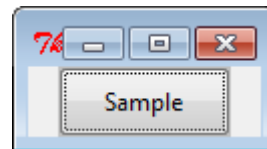
```
from tkinter import ttk  
import tkinter
```

```
root = tkinter.Tk()
```

```
ttk.Style().configure("TButton", padding=6, relief="flat",  
background="#ccc")
```

```
btn = ttk.Button(text="Sample")  
btn.pack()
```

```
root.mainloop()
```



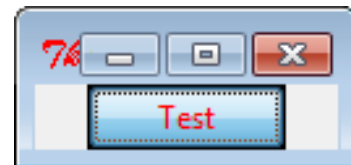
```
import tkinter
from tkinter import ttk
```

```
root = tkinter.Tk()
```

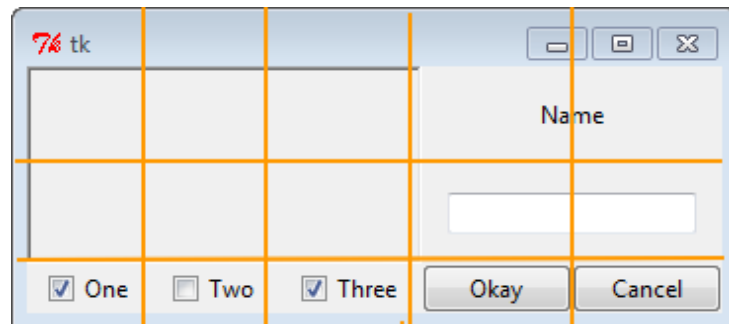
```
style = ttk.Style()
style.map("C.TButton",
        foreground=[('pressed', 'red'), ('active', 'blue')],
        background=[('pressed', '!disabled', 'black'),
                    ('active', 'white')]
        )
```

```
colored_btn = ttk.Button(text="Test", style="C.TButton").pack()
```

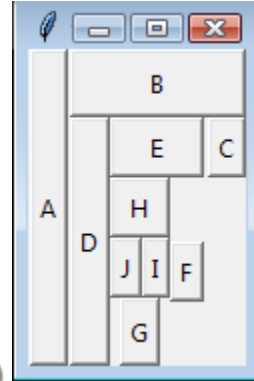
```
root.mainloop()
```



```
from tkinter import *
from tkinter import ttk
root = Tk()
content = ttk.Frame(root)
frame = ttk.Frame(content, borderwidth=5, relief="sunken", width=200, height=100)
namelbl = ttk.Label(content, text="Name")
name = ttk.Entry(content)
onevar = BooleanVar()
twovar = BooleanVar()
threevar = BooleanVar()
onevar.set(True)
twovar.set(False)
threevar.set(True)
one = ttk.Checkbutton(content, text="One", variable=onevar, onvalue=True)
two = ttk.Checkbutton(content, text="Two", variable=twovar, onvalue=True)
three = ttk.Checkbutton(content, text="Three", variable=threevar, onvalue=True)
ok = ttk.Button(content, text="Okay")
cancel = ttk.Button(content, text="Cancel")
content.grid(column=0, row=0)
frame.grid(column=0, row=0, columnspan=3, rowspan=2)
namelbl.grid(column=3, row=0, columnspan=2)
name.grid(column=3, row=1, columnspan=2)
one.grid(column=0, row=3)
two.grid(column=1, row=3)
three.grid(column=2, row=3)
ok.grid(column=3, row=3)
cancel.grid(column=4, row=3)
root.mainloop()
```

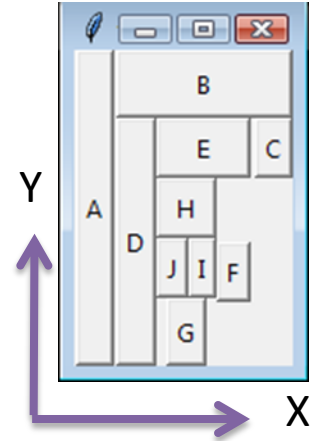


```
from tkinter import *  
root = Tk()  
Button(root, text="A").pack(side=LEFT, expand=YES, fill=Y)  
Button(root, text="B").pack(side=TOP, expand=YES, fill=BOTH)  
Button(root, text="C").pack(side=RIGHT, expand=YES, fill=NONE,  
anchor=NE)  
Button(root, text="D").pack(side=LEFT, expand=NO, fill=Y)  
Button(root, text="E").pack(side=TOP, expand=NO, fill=BOTH)  
Button(root, text="F").pack(side=RIGHT, expand=NO, fill=NONE)  
Button(root, text="G").pack(side=BOTTOM, expand=YES, fill=Y)  
Button(root, text="H").pack(side=TOP, expand=NO, fill=BOTH)  
Button(root, text="I").pack(side=RIGHT, expand=NO)  
Button(root, text="J").pack(anchor=SE)  
root.mainloop()
```



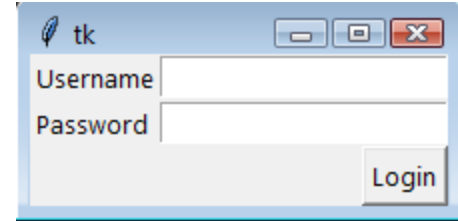
GUI : Pack Geometry Manager

- When you insert button A in the root frame, it captures the left-most area of the frame, it expands, and fills the Y dimension.
- When you insert the next button, B, into the root window, it picks up space from the remaining area but aligns itself to TOP, expand-fills the available area, and fills both X and Y coordinates of the available space.
- The third button, C, adjusts to the right-hand side of the remaining space.
- The anchor attribute used in some lines provides a means to position a widget relative to a reference point. If the anchor attribute is not specified, the pack manager places the widget in the center of the available space or the packing box.



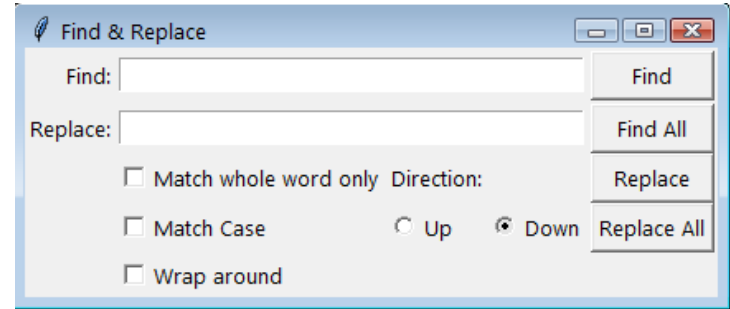
GUI : Grid Geometry Manager

```
from tkinter import *  
root = Tk()  
Label(root, text="Username").grid(row=0, sticky=W)  
Label(root, text="Password").grid(row=1, sticky=W)  
Entry(root).grid(row=0, column=1, sticky=E)  
Entry(root).grid(row=1, column=1, sticky=E)  
Button(root, text="Login").grid(row=2, column=1, sticky=E)  
root.mainloop()
```



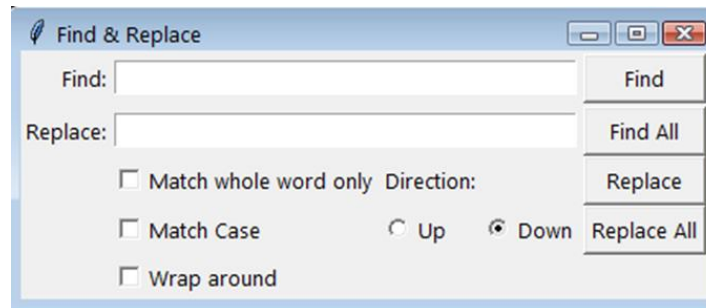
- the grid position defined in terms of rows and column positions for an imaginary grid table spanning the entire frame.
- The width of each column (or height of each row) is automatically decided by the height or width of the widgets contained in the cell.
- You can use the argument `sticky=N+S+E+W` to make the widget expandable to fill the entire cell of the grid.

```
from tkinter import *
top = Tk()
top.title('Find & Replace')
Label(top, text="Find:").grid(row=0, column=0, sticky='e')
Entry(top).grid(row=0, column=1, padx=2, pady=2, sticky='we', columnspan=9)
Label(top, text="Replace:").grid(row=1, column=0, sticky='e')
Entry(top).grid(row=1, column=1, padx=2, pady=2, sticky='we', columnspan=9)
Button(top, text="Find").grid(row=0, column=10, sticky='ew', padx=2, pady=2)
Button(top, text="Find All").grid(row=1, column=10, sticky='ew', padx=2)
Button(top, text="Replace").grid(row=2, column=10, sticky='ew', padx=2)
Button(top, text="Replace All").grid(row=3, column=10, sticky='ew', padx=2)
Checkbutton(top, text='Match whole word only').grid(row =2, column=1, columnspan=4,
sticky='w')
Checkbutton(top, text='Match Case').grid(row =3, column=1, columnspan=4, sticky='w')
Checkbutton(top, text='Wrap around').grid(row =4, column=1, columnspan=4, sticky='w')
Label(top, text="Direction:").grid(row=2, column=6, sticky='w')
Radiobutton(top, text='Up', value=1).grid(row=3, column=6, columnspan=6, sticky='w')
Radiobutton(top, text='Down', value=2).grid(row=3, column=7, columnspan=2, sticky='e')
top.mainloop()
```



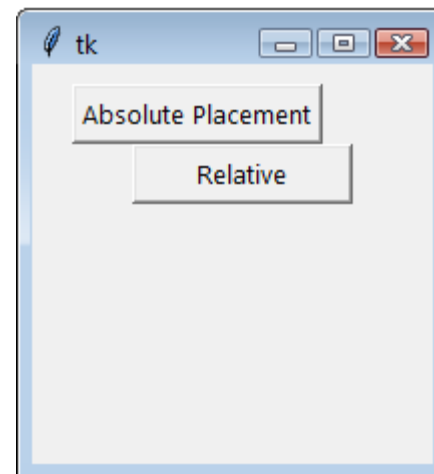
GUI : Grid Geometry Manager

- Notice how just 14 lines of core grid manager code generates a complex layout such as the one shown in the following screenshot. In contrast, developing this with the pack manager would have been much more tedious.
- Another grid option that you can sometimes use is the `widget.grid_forget()` method. This method can be used to hide the widget from the screen. When you use this option, the widget exists in its place but becomes invisible. The hidden widget may be made visible again but any grid options that you had originally assigned to the widget will be lost.



GUI : Place Geometry Manager

```
from tkinter import *  
root = Tk()  
# Absolute positioning  
Button(root, text="Absolute Placement").place(x=20, y=10)  
# Relative positioning  
Button(root, text="Relative").place(relx=0.8, rely=0.2,  
relwidth=0.5, width=10, anchor = NE)  
root.mainloop()
```



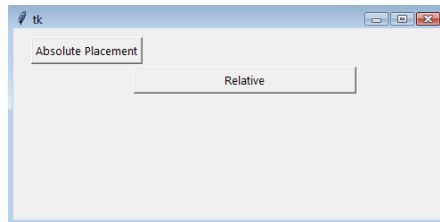


- The place geometry manager is **the most rarely used** geometry manager in tkinter.
- The important options for place geometry include:

Absolute positioning (specified in terms of $x=N$ or $y=N$)

Relative positioning (key options include `relx`, `rely`, `relwidth`, and `relheight`)

You may not see much of a difference between absolute and relative positions simply by looking at the code or the window frame. If, however, you try resizing the window, you will notice that the button placed absolutely does not change its coordinates, while the relative button changes its coordinates and size to fit the new size of the root window.



Tkinter provides three modules that can create **pop-up** dialog windows for you

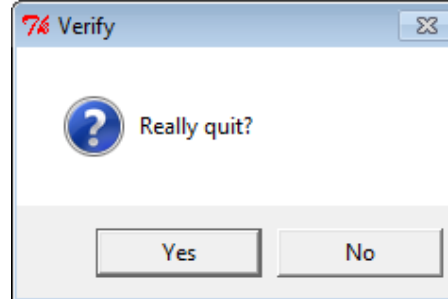
The **tkinter.messagebox** dialogs module, provides an assortment of common popups for simple tasks.

The **tkinter.filedialog**, allows the user to browse for files.

The **tkinter.colorchooser** module, allows the user to select a color.

GUI : Using Module Tkinter

Tkinter provides a set of dialogues, which can be used to display message boxes, showing warning or errors, or widgets to select files and colours. There are also simple dialogues, asking the user to enter string, integers or float numbers.

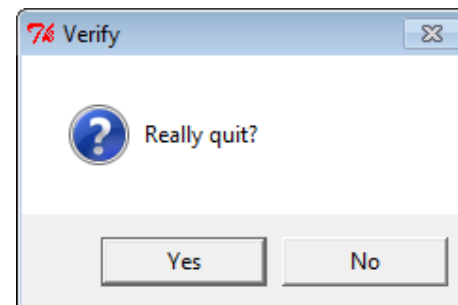
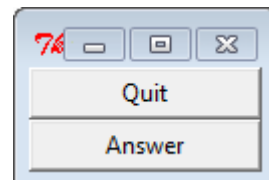



```
from tkinter import *
from tkinter.messagebox import *

def answer():
    showerror("Answer", "Sorry, no answer available")

def callback():
    if askyesno('Verify', 'Really quit?'):
        showwarning('Yes', 'Not yet implemented')
    else:
        showinfo('No', 'Quit has been cancelled')

Button(text='Quit', command=callback).pack(fill=X)
Button(text='Answer', command=answer).pack(fill=X)
mainloop()
```





- `askokcancel(title=None, message=None, **options)`
Ask if operation should proceed; return true if the answer is ok
- `askquestion(title=None, message=None, **options)`
Ask a question
- `askretrycancel(title=None, message=None, **options)`
Ask if operation should be retried; return true if the answer is yes
- `askyesno(title=None, message=None, **options)`
Ask a question; return true if the answer is yes
- `askyesnocancel(title=None, message=None, **options)`
Ask a question; return true if the answer is yes, None if cancelled.
- `showerror(title=None, message=None, **options)`
Show an error message
- `showinfo(title=None, message=None, **options)`
Show an info message
- `showwarning(title=None, message=None, **options)`
Show a warning message

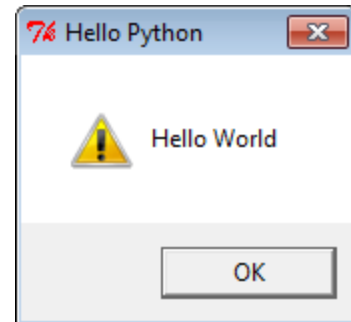
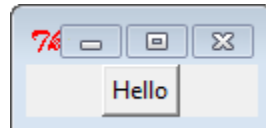
```
import tkinter
import tkinter.messagebox
```

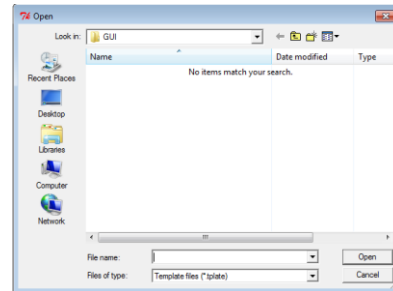
```
window = tkinter.Tk()
```

```
def helloCallBack():
    tkinter.messagebox.showwarning( "Hello Python", "Hello
World")
```

```
button = tkinter.Button(window, text = "Hello", command =
helloCallBack)
```

```
button.pack()
window.mainloop()
```





```
from tkinter import *
from tkinter.filedialog import askopenfilename
from tkinter.messagebox import showerror

def callback():
    fname = askopenfilename(filetypes=(("Template files", "*.tplate"),
                                       ("HTML files", "*.html;*.htm"),
                                       ("All files", "*.*") ))

    if fname:
        try:
            print(fname)
        except:
            showerror("Open Source File", "Failed to read file\n'%s'" % fname)
    return

Button(text='File Open', command=callback).pack(fill=X)
mainloop()
```

```
from tkinter import *  
from tkinter.colorchooser import *  
  
mGui = Tk()  
mGui.geometry("600x300+500+500")  
mGui.title("Hexadecimal Color Chooser")
```

A geometry string is a standard way of describing the size and location of a top-level window on a desktop.

'wxh±x±y'

The **w** and **h** parts give the window width and height in pixels. They are separated by the character 'x'. **+x** and **±y**, specify that the left and top sides of the window should be x and y pixels from the left side of the desktop.

```
# tkinter.colorchooser.askcolor returns a two-item tuple like  
# this: ((128.5, 64.25, 64.25), '#804040')  
def getColor():  
    color_choice = colorchooser.askcolor()[1] # get the hex code  
    color = Label(mGui, bg=color_choice)  
    color.pack()  
    hexcode = Label(mGui, text="The hexadecimal color code is: " + color_choice)  
    hexcode.pack()  
  
button = Button(mGui, text="Choose a color", command = getColor)  
button.place(x=0, y=0)  
mGui.mainloop()
```

Class Work

Design a GUI for a Calculator

