

tkinter Toys Starter Set

This is a limited reference to help students get started with the tkinter GUI module in Python. It assumes a

sound beginner's knowledge of Python, the most current versions of Python and tkinter (NOT Tkinter with a capital T), a basic orientation to object oriented programming, a Windows 10 computer with Python and IDLE installed. If anything said so far gives you the slightest reason to pause, go here: <https://www.python.org/about/gettingstarted/>. You should know that tkinter replaced Tkinter in Python 3.0. and the next "level" of tkinter, **tkk**, is not addressed here except to say it replaces some, maintains some, and adds to, tkinter.

Importing or "Loading" tkinter: of the two valid ways to import tkinter, this doc assumes we use: **from tkinter import *** [Alternative 2, *import tkinter as tk*, requires prefixing commands with "tk." which confuses new users.]

A general idea of the process for the creation of a GUI object oriented application might be as follows:

- 1: Import tkinter
- 2: Establish a root window
- 3: Define the root geometry
- 4: Set up needed variables
- 5: Plan and build event functions
- 6: Instantiate and define widgets & set initial focus
- 7: Setup manual bindings as necessary
- 8: Deploy widgets with a geometry manager
- 9: Program app code
- 10: Establish mainloop boundry

Quick Start: For examples and use in testing as you learn, here is a working quick start gui framework.

```
from tkinter import *
root=Tk()
root.wm_attributes("-fullscreen", True)
root.configure(
background='snow3')
```

```
def egress():
    root.destroy()
```

```
def textInput(event):
    e1.delete(0,END)
    e1.unbind("<Key>")
    e1.bind("<Return>", useInput)
    e1.insert(0, e1.get())
```

```
def useInput(self):
    L1Text.set(e1.get())
    e1.delete(0,END)
    e1.insert(0, "Line "+ str(Counter.get()) + ": ")
    e1.bind("<Key>", textInput)
    Counter.set(Counter.get()+1)
```

```
top1 = Toplevel(root, bg="light blue")
top1.geometry('800'+ 'x' + '400')
top1.title("top1 Window: Starter Set Test Layout")
top1.attributes("-topmost", 1)
```

```
bexit=Button(root, command=egress, text="Close\rButton", \
bg="brown",fg="white",\
font=("Calibri 14 bold"), width=10, height=2)
bexit.pack(ipady=2, ipadx=2, pady=3, padx=3,side="left",
anchor="nw")
```

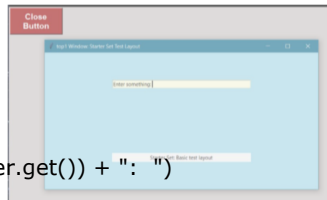
```
EnterPrompt=StringVar()
akey=StringVar()
L1Text=StringVar()
Counter=IntVar()
```

```
EnterPrompt.set("Enter something: ")
akey.set("")
L1Text.set("")
Counter.set(2)
```

```
e1=Entry(top1, width=50, bg="beige",
textvariable=EnterPrompt)
e1.bind("<Key>", textInput)
e1.focus_set()
e1.icursor(END)
e1.pack(pady=80)
```

```
L1Text.set("Starter Set: Basic test layout")
l1=Label(top1, textvariable=L1Text, width= 50)
l1.pack(anchor= "center", side="bottom", pady=80)
mainloop()
```

This code with extensive documentation can be found (and downloaded) at www.wikipython.com Look for it close to where you will find TB5.



tkinter Widgets
In this Quick Start app:
root Toplevel
Button Label
Entry

Others are:
Canvas Checkbutton
Frame Labelframe
Listbox Menubutton
Message Panedwindow
Scale Radiobutton
Scrollbar Spinbox
Text Messagebox

Vocabulary: note: **w** is a widget **instance**; **↵** = yields

WIDGET an object that you use to input, manipulate and display information. There are 18 widgets, this starter set refers to only **root** (not actually a widget), **toplevel**, **button**, **label** and **text**. Widgets have **attributes** (values) and **methods** (actions); usually some attributes are assigned values when creating the widget.

ATTRIBUTES are characteristics of an object like padding, colors, and sizes; sometimes called **OPTIONS** -changeable

METHODS, sometimes called **COMMANDS**, are actions that an object can take if programmatically called.

A **CALLBACK** is a function that is "called" or executed when an **event** that as been **bound** to a widget occurs. (a subroutine)

BIND means to link a **callback function** (code that does something) to a widget **instance** so that when a specific **event** occurs, that code will be executed. Some widgets have **innate bindings**, for example, a left mouse click on a button is a built-in event. You **bind** it to a **callback** with "command=" when you create that button **instance**. Some events require that you create a manual binding to connect them to a widget **instance**. For example, if you want something special to happen when a user right-clicks a label widget, you must specially create that binding.

INSTANCE: tkinter supports 18 classes of widgets, but a class is just a general blueprint, to create an object you can use you must create it by telling tkinter how to make your special object (your **instance**) from a general class.

EVENT: something that occurs which might

continued on page 2

Deploy your widgets - w.pack (widget, attributes) Use one of the 3 geometry managers to place and make widgets visible. The 3 managers are: **pack** - a mode ideally suited for learning or very simple GUI interfaces; **grid** - an easy mode that works well for most GUI situations; grid works on cols and rows. **place** - a complex, precise, flexible system not covered here.

Methods: Universal Methods (x=widget name.a geometry name)

x_forget() remove from manager but do not destroy, can reuse **ex:** label1.grid_forget(), retrieve it by repeating the original grid command
x_info() ↵ a dictionary of options **ex:** print(label1.pack_info())
x_slaves() returns list of sub widgets as tkinter widget references
x_configure(opts) same as .pack()

Geometry Specific Methods

place: has no other Methods.

pack and grid: **x_propagate(flag)** ; True/False; enables resizing of child widgets if too small

grid:

w.grid_bbox(column=None, row=None, col2=None, row2=None)

w.grid_size() tuple with number of columns and rows

w.grid_location(x,y) ↵ a tuple with indexes

w.grid_remove() removes widget from mgr; available for reuse

To change the following, you must call these on a widget's **parent**:

grid_columnconfigure(index, options)

grid_rowconfigure(index, options)

index = column number

options: minsize=, pad=, weight=

Options and Attributes: for configure: option: default: value: comment

pack - attributes for **configure()**
anchor= center/top: compass points:
expand= false : 0,1 : fill extra space
fill= None : X (fill horiz), Y fill
vert, BOTH: fill all space

To make a widget fill the entire master widget, set **fill=** to BOTH and **expand=** to a non-zero value.

in_ = w pack inside w

ipadx= 0 : int : internal pad horiz

ipady= 0 : int : internal pad vert

padx= 0 : external pad horiz

pady= 0 : external pad vert

side= "top" : "left", "right", "top"

"bottom", : side to pack against, can mix sides in one geometry manager

grid - attributes for **configure()**
column= 0 : int : starts with 0
columnspan= 1 : int : span columns
in_ = w parent : sibling w : place w in w

ipadx= 0 : int : internal padding hz

ipady= 0 : int : internal padding vt

padx= 0 : int : external padding hz

pady= 0 : int : external padding vt

row= first empty : row num :

rows start with 0

rowspan= 1 : int : span multiple rows

sticky= centered : Compass Points :

strings or contents, W+E stretch horz, W+E+N+S

fills all : alignment



Selected tkinter Widget Options or Attributes	Button	Entry	Label	Toplevel	Value Type/Default
activebackground	•		•		color / system
activeforeground	•		•		color / system
anchor	•		•		compass points or "center" / usually center
background or -bg	•	•	•		color / system
bitmap	•		•		"" or filename / (see list)
borderwidth or -bd	•	•	•	•	distance value / 2 pixels
command	•				callback function name / none
container				•	boolean / false
cursor	•	•	•	•	cursor name / system
default	•				normal, active, disabled /
disabledforeground	•	•	•		color / n/a
font	•	•	•		font 3 tuple /
foreground or -fg	•	•	•		color /
height	•		•	•	text-lines, image-distance
highlightbackground	•	•	•	•	color / system
highlightcolor	•	•	•	•	color / system
highlightthickness	•	•	•	•	distance value / 1 pixel
image	•		•		bit file: gif, pgm, ppm /
justify	•	•	•		see justify choices / center
padx	•		•	•	distance value / 1 pixel
pady	•		•	•	distance value / 1 pixel
relief	•	•	•	•	see relief choices / "sunken"
repeatdelay	•				milliseconds before engage /
repeatinterval	•				miliseconds between execution
state	•	•	•		Normal or Disabled / normal
takefocus	•	•	•	•	0 or 1 / 1
text	•		•		a string /
textvariable	•	•	•		a string / (set to control var)
underline	•		•		integer /
width	•	•	•	•	characters /
wrlength	•		•		distance value / nowrap
STARTER SET METHODS					INDICES:
TOPLEVEL / LABEL / BUTTON					number character -0 1st point
cget (option)					end char after string
configure (options/values)					insert next after cursor
BUTTON					sel.first 1st char
flash					sel.last char after select
invoke callback					@number x coord
ENTRY					
bbox index					
cget option					
configure option(s)/value(s)					
delete first, last					
get					
icursor index					
index index					
insert index string					
scan					
mark x					
dragto x					
selection					
adjust index					
clear index					
from index					
present start,end					
range index					
validate					
xview					
2 real fractions visible span to left edge fraction number, what what=units or pages					

PRIMARY BINDINGS

<Button1> : leftmost : <1> is alias
 <Button2> : middle if available
 <Button3> : right-most mouse button

<ButtonRelease1> :
 <Leave> : mouse pointer left widget
 <B1-Motion> : movement w/ button held down

<DoubleButton1> : double click
 <Enter> : mouse pointer in widget
 <FocusIn> : keyboard focus moved to widget
 <FocusOut> : keyboard focus moved away from widget

<Return> : the keyboard enter key
 <Key> : w.bind("<Key>", callback) any keypress
 "x" : any letter : ex: label.bind("x", callback)
 Special Keys: Cancel (Break),

BackSpace, Tab, Shift, Contol, Alt, Pause, Escape, Page Up Page Down..

Event Object passed to **callback** includes:
 widget - tkinter instance
 x,y - current mouse position
 x_root, y_root - mouse position relative to the upper left corner of the screen, in pixels.
 char - character code (keyboard events only), as a string.
 keysym - key symbol (keyboard events)
 keycode - the key code (keyboard events)
 num - The button number (mouse button events only).
 width, height - new widget size, in pixels
 type - event type

Vocabulary: continued from page 1

cause your program or application to interact with the user and/or data; for example a mouse button press or a <RETURN> keypress .

Compass points: 'n', 'ne', 'e', 'se', 's', 'sw', 'w', 'nw', 'center'. Also constants W,E,N,S. W+E to stretch. "wens" to fill all

Colors: can be given as the names of colors in the rgb.txt file (downloaded with tkinter); also hex definitions #rbg, #rrggbb, or #rrrggbbb

Distance: Pixels → numeric; absolute distances → strings with a trailing character denoting units: c-centimeters, i-inches, m-millimeters, p- printer's points - these vary with font used.

Fonts: Ex: font=("Verdana 10 bold"). Font sizes with positive numbers measured in points; sizes with neg numbers are measured in pixels. Note font definition is in quotes.

Justify: "left", "center", "right", "fill" include quotes

Region: 4 space-delimited legal distances ex: "3i 2i 4.5i 2i"

Relief: "raised", "sunken", "flat", "groove", "ridge"

RAISED SUNKEN FLAT GROOVE RIDGE

Wrap: "none", "char", "word"

Cursors: many available such as: "arrow" "circle" "clock" "cross" "dotbox" "exchange" "fleur" "heart" "man" "plus" "shuttle" "watch"

Bitmaps: 'error', 'gray75', 'gray50', 'gray25', 'gray12', 'hourglass', 'info', 'questhead', 'question', 'warning'

Images: B&W id constructor: myBWpic = tk.BitmapImage (file=myimagefile.xbm); Color: myphotoimage = PhotoImage (file=myimagefile. { .gif, .pgm, .ppm formats })

Control Variables: StringVar(), DoubleVar(), BooleanVar(), IntVar(); use .set() and .get() , tracked and auto-changed.

Starter Set: Window Information Methods	Starter Set: Window Manager Methods
ex: top1.wininfo_geometry()	attributes ex: root.attributes("-fullscreen", True)
height() : widget height in pixels	-alpha transparency, 0.0-1.0
rootx() : left edge cood rel to screen	-fullscreen NEW
rooty() : left edge cood rel to screen	-topmost place this window on top
screenheight() : height of widget screen	-disabled disables window
screenwidth() : width of screen in pixels	-transparentcolor trans color index of toplvl
vrootx() : x offset of virtual root rel to root win	-transparent window area transparent
vrooty() : y offset of virtual root rel to root win	geometry widthxheight+woffset+yoffset
width() : widget width, pixels (update_idletasks)	command (value=); WM_COMMAND
x() : upper corner coord	iconify iconify widget
y() : upper corner coord	withdraw unmap w; deiconify to remap
pathname(displayof=0) : full window name	deiconify display, set focus, raise

.mainloop() - This method **must** be called - generally after all the static widgets are created - to start processing events. You can leave the main loop with the .quit() method. You can also call this method inside an event handler to resume the main loop.

Criticism & Comment Appreciated:
 john@johnaakey.com www.wikipython.com
 No warranty made for the accuracy of this document but we try! Happy coding!