Big Daddy's PYTHON TOOLBOX For 3.5 for tkinter toys

R1 V052217B

Step 1: import tkinter (as tk) optional
from tkinter import * (if you were going to deploy ttk you would add **from tkinter import ttk**)

!Last step: tkinter.mainloop() forget it and absolutely nothing will happen, at all

Step 2: establish a **root** window
root = tkinter.Tk()
Consider

Suggestion-begin by getting screen dimensions:
sW = root.winfo_screenwidth()
sH = root.winfo_screenheight()
Make these the next lines after setting root.

tkinter is vast - this is a VERY limited treatment to help get you started or remind you of what you already know. tkinter replaced Tkinter in Python 3.0. tkk replaces some tkinter command, leaves some in place, adds others. tix adds compound widgets. Please see **www.wikipython.com** for more on tkinter

Step 3: define **root** geometry *not required but recommended
root.geometry(str(sW) + "x" + str(sH))

can define any pixel dimensions
this example grabs whole monitor
root's geometry is **defined**, not **set** by pack, gird, or place- like a widget

Vocabulary: In this document **ATTRIBUTES** are fixed but changeable characteristics like fonts, colors, sizes; in most tkinter docs these are called **OPTIONS** which are confused with **METHODS** which are actions that an object can take if programmatically called; ***w*** is any widget instance; callback means the function bound/called to respond to a specific event, such as a key press or a mouse click.

Step 4: set up **variables**: You will probable need lots of variables but be aware of one in particular. Several widgets allow you to define a textvariable when they are created, (textvariable=*myString*). Before that creation event you need to have defined a string, like *myString* using StringVar(), i.e., *myString*= StringVar() which is usually followed by assigning a value to it with .set ,i.e., ***myString*.set("some default text")**

(5B) Suggested new step: Toplevel - consider creating at least one **Toplevel** with maximized **root** as its **parent**. (1) With two Toplevels, screens can alternate with .lift or .focus_set() methods. (2) You can size Toplevel to frame an unknow monitor's resolution while working in a known area. (97% of all current laptops can display **1024 x 768** - consider that as a central working area.) Toplevel's geometry is defined like root's BUT you have to initially bring up a Toplevel window (for ex: "**top1**") with the command: **top1**.wm_attributes("-topmost", **True**) and later remove it with wm_attributes("-topmost", **False**) before moving focus to a new window.

Levels of Event Bindings:
Instance: bind an event to a specific widget using the .bind() method. For example see below - in that case there is no need for a ***widget*.bind(event)** statement because "clicking" is inherient to the button widget.
Class: bind all widgets in a class with the .bind_class() method. Example: self.bind_class(w_type, '<Button-2>', self.__callback)
Application: Event calls a handler regardless of what widget has focus using the .bind_all() method. Example:
self.bind_all('<Key-Print>', self.__printScreen)
Toplevel: a Toplevel or root window can also apply the bind command.

Step 5: event functions - plan/build with at least placeholder structures. You can finsh them later.

Step 6: define widgets - set initial attribute values, focus status, and connect event functions as needed. *button clicks do not need a binding, just set **command=yourfunction**, syntax like this

wName= tkinter.widget_type(attributes) Example: **but1=tk.Button(top1, command= myb1function, bg='light blue', text='Push Me')**

Step 7: set bindings (as needed) - a binding links an event, like a mouse click or key-press, to a function containing your **callback** response code. There are many bindings (see above & below) 2 main groups: keyboard and mouse; 2 examples:
w.bind("<Button-1>", callback) *<-note quotes*
w.bind("<Return>", callback)

Widget Name	**tkinter/ttk**
CONTAINERS	
Toplevel	tkinter
Frame	tk/ttk repl
LabelFrame	tk/ttk repl
Canvas	tkinter
PanedWindow	tk/ttk repl
BUTTONS	
Button	tk/ttk repl
Checkbutton	tk/ttk repl
Radiobutton	tk/ttk repl
Menubutton	tk/ttk repl
SELECTION	
Scale	tk/ttk repl
Scrollbar	tk/ttk repl
Spinbox	tkinter
Combobox	new ttk
COMMUNICATION	
Entry	tk/ttk repl
Label	tk/ttk repl
Text	tkinter
Listbox	tkinter
Message	tkinter
messagebox	tkinter
Notebook	new ttk
STRUCTURAL COMPONENTS	
Progressbar	new ttk
Sizegrip	new ttk
Separator	new ttk
Treeview	new ttk

In the chart above, if you load ttk, the tkinter widgets labeled "tk/ttk repl" are replaced by themed ttk widgets-which have different options. ttk also **adds** the widgets shaded in light grey labeled "new ttk". The widgets which say "tkinter" are unaffected and processed by the original tkinter code.
See back for *w* OPTIONS

Step 8: deploy your widgets - call on one of the 3 "geometry" managers to make your widget visible where and how you want it. These are the three geometry managers:
PACK - a mode ideally suited for learning or very simple GUI interfaces; w.pack(attributes and methods)
GRID - an easy to implement mode that works well for most GUI situations; works on cols and rows - starting with 0 not 1
PLACE - a precise, complex, flexible system for extensive complicated interfaces; placement down to the pixel.

(9) The **Last** step: tkinter.mainloop()
don't forget .mainloop() or absolutely nothing will happen, at all

Attributes (options) common to ALL Geometries: none
Methods common to all Geometries:
x_forget() remove from manager but do not destroy, can reuse
x_info() return dictionary of options
x_slaves() returns list of sub widgets as tkinter widget references
x_configure(options) *see below*

Geometry Compass Points: 'n', 's', 'e', 'w', 'ne', 'nw', 'se', 'sw'; a default may be centered which is not a programable option. Note lower case in quotes.
Propagation: If enabled (default), manager trys to change widget size if child widget changes size.
Distance: **c**=centimeters, **i**=inches, **m**=millimeters, **p**= printer's points (1/72"), **none** pixels

Pack - attributes for **configure()**

OPTION	**Default: Options : Comment**
anchor=	CENTER : 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", "bottom", "top" : side to pack against, can mix sides in one geometry manager

OTHER METHODS:
pack_propagate(flag) : True = propagation

Place - attributes for **configure()**

OPTION	**Default: Options : Comment**
anchor=	NW : compass points :
bordermode=	INSIDE : INSIDE/OUT-SIDE : inside parents border
height=	none : int : in pixels
In_=*w*	pack inside w
relheight=	none : O.O to 1.0 : fraction of parent, vert
relwidth=	none : O.O to 1.0 : fraction of parent, horiz
relx=	none : O.O to 1.0 : offset fraction of parent, horiz
rely=	none : O.O to 1.0 : offset fraction of parent, vert
width=	none : int : in pixels
x=	0 : int : horiz offset in pixels
y=	0 : int : vert offset in pixels

OTHER METHODS:
None

Grid - attributes for **configure()**

OPTION	**Default: Options : Comment**
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 : W+E stretch horz, W + E + N + S alldir : alignment

OTHER METHODS:
pack_propagate(flag) : True = propagation
grid_bbox(column=None, row=None, col2=None, row2=None)
grid_size() : tuple of # of col and rows
grid_location(x, y) : returns tuple w/ indexes
grid_remove() : remove w from mgr, reuse
To change the following, you must call these on widget's parent:
grid_columnconfigure(index, options)
grid_rowconfigure(index, options)
Index options: Minsize=, pad=, weight=

PRIMARY BINDINGS
<Button1> : leftmost : <1> is alias
<Button2> : middle if available
<Button3> right-most mouse button :
<ButtonRelease1> :
<Leave> : mouse pointer left widget
<B1Motion> : movement with button down
<DoubleButton1> : double click
<Enter> : mouse pointer entered widget
<FocusIn> : keyboard focus moved to w
<FocusOut> : keyboard focus moved away
<Return> : the keyboard enter key
<Key> : w.bind("<Key>",key) any keypress
"X" : a letter : ex: frame.bind("H", callback)

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 (Configure events).
type - event type

Big Daddy's PYTHON TOOLBOX For 3.5

R1

V052217B

Primary Widget Attributes

Primary Widget Attributes	Button	Checkbutton	Entry	Radiobutton	Scale	Menubutton	Label	LabelFrame	Text	Spinbox	Listbox	Message	Toplevel	Canvas	Frame	Scrollbar	PanedWindow	messagebox	count
bd | borderwidth	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		17
bg | background	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		17
cursor	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		17
relief	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		17
width	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		17
highlightbackground	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X			16
highlightcolor	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X			16
highlightthickness	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X			16
takefocus	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X			16
fg | foreground	X	X	X	X	X	X	X	X	X	X	X	X							12
font	X	X	X	X	X	X	X	X	X	X	X	X							12
height	X	X		X		X	X	X	X		X		X	X	X		X		12
state	X	X	X	X	X	X	X		X	X	X			X					11
padx	X	X		X		X	X	X	X			X	X		X				10
pady	X	X		X		X	X	X	X			X	X		X				10
activebackground	X	X		X	X	X	X			X						X			8
disabledforeground	X	X	X	X		X	X			X	X								8
justify	X	X	X	X		X	X			X		X							8
textvariable	X	X	X	X		X	X			X		X							8
text	X	X		X		X	X	X				X							7
anchor	X	X		X		X	X					X							6
command	X	X		X	X					X						X			6
activeforeground	X	X		X		X	X												5
bitmap	X	X		X		X	X												5
compound	X	X		X		X	X												5
image	X	X		X		X	X												5
selectbackground			X						X	X	X			X					5
selectborderwidth			X						X	X	X			X					5
selectforeground			X						X	X	X			X					5
underline	X	X		X		X	X												5
wraplength	X	X		X		X	X												5
xscrollcommand			X						X	X	X			X					5
exportselection			X						X	X	X								4
insertbackground			X						X	X				X					4
insertborderwidth			X						X	X				X					4
insertofftime			X						X	X				X					4
insertontime			X						X	X				X					4
insertwidth			X						X	X				X					4
repeatdelay	X				X					X						X			4
repeatinterval	X				X					X						X			4
class								X					X		X				3
colormap								X					X		X				3
container								X					X		X				3
indicatoron		X		X		X													3
orient					X											X	X		3
overrelief	X	X		X															3
variable		X		X	X														3
visual								X					X		X				3
yscrollcommand									X		X			X					3

Your Notes Here:

This table of 49 widget attributes represents 79.7% of all widget options. There are an additional 79 attributes which apply to only 1 or 2 widgets each. The entire table is available on wikipython.com

SPECIAL KEY BINDINGS

Special keys are Cancel (the Break key), BackSpace, Tab, Return (the Enter key), Shift_L (any Shift key), Control_L (any Control key), Alt_L (any Alt key), Pause, Caps_Lock, Escape, Prior (Page Up), Next (Page Down), End, Home, Left, Up, Right, Down, Print, Insert, Delete, F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, Num_Lock Scroll_Lock.

Protocols: work like event bindings

WM_DELETE_WINDOW controls events when user closes window:
w.protocol(**"WM_DELETE TE _WINDOW", callback)**
Also:
WM_TAKE_FOCUS
WM_SAVE_YOURSELF

A few useful color names
`white', `black', `red', `green', `blue', `cyan', `yellow', `magenta','snow', 'ghost white', 'white smoke', 'slate blue', 'medium slate blue', 'light slate blue', 'lawn green', 'medium spring green', 'green yellow', 'indian red', 'saddle brown', 'pale violet red', 'maroon', 'medium violet red', 'violet red', 'medium orchid', 'dark orchid', 'dark violet', 'blue violet', 'gray10', 'gray20', 'gray30', gray 99, 'slate gray', 'light slate gray', `gray', 'light grey', 'midnight blue', 'navy', 'cornflower blue', 'light coral', 'tomato', 'orange red', 'red', 'hot pink', 'deep pink', 'pink', 'light pink', 'DodgerBlue4', 'SteelBlue1', 'SteelBlue4', DeepSkyBlue2', 'CadetBlue4', 'turquoise1', 'turquoise4', 'cyan4', 'DarkSlateGray1', 'PaleGreen4', 'DarkOliveGreen1'

Criticism & Comment appreciated:
john@johnoakey.com
www.wikipython.com
No warranty is made as to the accuracy of this information.
Hey, this is only Revision 1.