BIG DADDY'S PYTHON TOOLBOX For 3.5

R2D

© 2017 John A. Oakey
V061917a

for tkinter toys

Step 1: import tkinter (as tk) optional
from tkinter import *
(if deploying 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 = tk.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 probably need lots of variables. In particular be aware of textvariable. Create this special variable object and set its value. ie, myStr=StringVar(); mystr.set("some default text") , then when creating a widget that has the textvariable attribute, just associate it: textvariable=myStr. Anytime mystr changes value, the text on the button, label, entry, spinbox, etc. will change automatically.

(5B) Suggested added step: Toplevel: consider creating at least one **Toplevel** with a maximized **root** as **parent**. With two, your screens can alternate with **.lift, .lower,** or **.focus_set()** methods and you can size Toplevel to frame an unknow monitor's resolution while working in a known area. (97% of all laptops can display **1024 x 768;** consider it 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 a command like: **top1.lift** (aboveThis=none) or top1.wm_attributes ("-topmost", **True**) and later remove it with wm_attributes("-topmost", **False**) or **.lower** 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 & at bottom) 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 ttk is loaded, the tkinter widgets labeled "tk/ttk repl" are replaced by themed ttk widgets-which have different options. ttk **adds** the widgets shaded light grey and labeled "new ttk". The widgets which say "tkinter" are unaffected and processed by the original tkinter code. **See back for *w* OPTIONS** and much more @ **www.wikipython.com**

Step 8: deploy your widgets - call on one of the 3 geometry managers to make your widget visible where and how you want it. The three geometry managers are:
PACK - a mode ideally suited for learning or very simple GUI interfaces; **w.pack** (or grid or place) **(widget, attributes and methods)**
GRID - an easy to implement mode that works well for most GUI situations; works on cols and rows - both start 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', 'center'; a default may be centered which may not be a programable option. Lower case & 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. Ex: "3i" or "10c"

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.

OPTION	Default: Options : Comment
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/OUTSIDE : 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

R2D

V061917a

This TOP 40 table of the 127 named widget options represent 349 or 74% of widget options reported by tkinter. An additional 123 attributes apply to 3 or fewer widgets each. **The entire table and the footnoes are available on :**

www.wikipython.com

	Primary Widget Attributes (or options)	bd | borderwidth	bg | background	cursor	relief	width	highlightbackground	highlightcolor	highlightthickness	takefocus	fg | foreground	font	height	state	padx	pady	activebackground	disabledforeground	justify	textvariable	text	anchor	command	activeforeground	bitmap	compound	image	selectbackground	selectborderwidth	selectforeground	underline	wraplength	xscrollcommand	exportselection	insertbackground	insertborderwidth	insertofftime	insertontime	insertwidth	repeatdelay	repeatinterval
	Values	+pixels	color	cursors *1	relief *2	characters *3	color	color	+pixels	0 or 1 or ""	color	font-3 tuple; name,	lines *3	NORMAL, DISABLED	+pixels	+pixels	color	color	left ,center, right	a string	a string	compass points	function name	color	"" or filename	LEFT,RIGHT,TOP,BOTTOM,	gif, pgm, ppm *5	color	+pixels	text color	integer	0, max line len int	float *4	1 or 0	color	+pixels	+milliseconds	+milliseconds	pixels	+milliseconds	+milliseconds
	Default	2 pix			"SUNKEN"				1	1		size, wt.		NORMAL	1p	1p						or center			*7	CENTER					0 is 1st	p,i,m,-	0 to 1	1	black	0	300	600	2		
Communication	messagebox												Messagebox has no options/attributes																												
	Message	●	●	●	●	●	●	●	●	●	●	●			●	●			●	●	●	●																			
	Listbox	●	●	●	●	●	●	●	●	●	●	●	●	●				●										●	●	●			●	●							
	Text	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●												●	●	●			●	●	●	●	●	●	●		
	Label	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●				●	●									
	Entry	●	●	●	●	●	●	●	●	●	●	●		●				●	●	●								●	●	●			●	●	●	●	●	●	●		
Select	Spinbox	●	●	●	●	●	●	●	●	●	●	●		●			●	●	●	●			●					●	●	●			●	●	●	●	●	●	●	●	●
	Scrollbar	●	●	●	●	●	●	●	●	●							●						●																	●	●
	Scale	●	●	●	●	●	●	●	●	●	●	●		●			●						●																	●	●
Buttons	Menubutton	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●				●	●									
	Radiobutton	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●				●	●									
	Checkbutton	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●				●	●									
	Button	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●				●	●								●	●
Containers	PanedWindow	●	●	●	●	●							●																												
	Canvas	●	●	●	●	●	●	●	●	●			●	●														●	●	●			●		●	●	●	●	●		
	LabelFrame	●	●	●	●	●	●	●	●	●	●	●	●		●	●					●																				
	Frame	●	●	●	●	●	●	●	●	●			●		●	●																									
	Toplevel	●	●	●	●	●	●	●	●	●			●		●	●																									

SPECIAL KEY BINDINGS

Special keys are Cancel (Break), BackSpace, Tab, Return(the Enter key), any Shift keyany Control key, 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_DELE TE_WINDOW", callback)** Also: **WM_TAKE_FOCUS**

Shipman reference: http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/index.html

A Few Basic Widget Methods

See a larger list on www.wikipython.com

.bind(event, function, add=None)	add=+ to activate multiple bindings
.bind_all (sequence=None, func=None,add =None)	applies to all widgets in the entire app
.bind_class (className, sequence=None, func=None, add=None)	bind all widgets in the entire class
.cget(option)	returns option value
.column_configure()	aplply to parent of grided widget
.configure(option=value, ...)	Learn before continuing; **see Shipman**
.destroy()	destroys w and all its children.
.focus_displayof()	↳ name of window with input focus, "none"
.focus_force()	forces input focus to w; "impolite"(?)
.focus_get()	returns w with focus or "none"
.focus_set()	occurs IF w's app has focus
.grab_current()	returns identifier or "none"
.grab_release()	release if grab in force
.grab_set()	grab all app events
.grab_set_global()	grab all events for entire screen
.grab_status()	local', 'global', 'none'
.grid_forget()	w disappears-not destroyed-forgets options
.grid_remove()	like forget but remembers options
.image_names()	returns all image names in app
.lift(aboveThis=None)	w window moved to top of the stack
.lower(belowThis=None)	w window moved to bottom of the stack

.mainloop()	*SEE NOTE
.option_clear()	resets options to default
.quit()	This method exits the main loop.
.rowconfigure()	grid management - call on the w parent
.selection_clear()	clear any selection w has
.selection_get()	returns selected text or if none tk.TclError
.tk_focusFollowsMouse()	force MOUSE focus versus keyboard
.tk_focusNext()	returns next w in normal sequence
.unbind(sequence, funcid=None)	removes event bind; remove funcid
.unbind_all(sequence)	remove all bindings for an event
.update()	forces display update; unpredictable;
.wait_variable(v)	local wait loop for v to be set; app cont
.winfo_fpixels(number)	↳ as float distance in pixels on w's display
.winfo_height()	↳ w height pixels; update idle tasks
.winfo_id()	an integer; needed for .winfo_pathname()
.winfo_pointerxy()	↳ tuple x,y per root or -1-1if mouse on different screen
.winfo_rootx()	returns left side x of w's root rel to parent
.winfo_rooty()	returns top side y of w's root rel to parent
.winfo_screenwidth()	width of screen in pixels
.winfo_width()	w in pixels; use .winfo_reqwidth() instead

.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@johnoakey.com
www.wikipython.com

No warranty is made as to the accuracy of this information. (But I have tried hard to get it right.) **Happy coding!**