© 2021 John A. Oakey V

tkinter 2021 Toolbox Reference **Section 1:** tkinter *without Ttk*

A Practical Outline of the Process of Creating a tkinter Application

(This creates a minimal environment for an actual application - some steps may differ depending on the needs or complexity of your design.)

- **Import:** **from tkinter import *** or **import tkinter as tk** - requires using **"tk."** prefix to commands)
- **Define** any variables needed to establish root
- **Establish root**: **root=Tk()** or root=tk.TK() a special singular widget which must be created to initialize tkinter (from now on in this doc, **import *** is assumed - no 'tk.' element needed) then ** **Define root geometry** and **option** values
- **Define** any variables needed for a Toplevel widget
- **Create a Toplevel widget** – a window in which to build your app; you can have more than one
- **Create Functions** and establish Variables **- Functions** can be **callback** widgets **invoked** to respond to **events**; **commands** and **lambda statements** can serve to respond as well
- **Create Widgets:** widget_name = widget_type**(**parent [,attribute values] [,options] [,a bound callback]**)**
- **Bind** additional or non-inherent **events**
- **Apply Geometry Placement**: with **grid, place** or **pack**
- **"root.mainloop()" -** "blocks", i.e., defines end of the program

Options, Values, Processes

Colors: can be given as names or as hex definitions: **#rgb, #rrggbb, #rrrgggbbb, #rrrrggggbbbb;** (see wikipython.com)
Control Variables - global, update automatically; constructors: =**DoubleVar()**, =**IntVar()**, =**BooleanVar(),** =**StringVar**, **;** use: v_name.**set(**value**)**and v_name.**get(**value**)**
Cursors: many available such as: "arrow" "circle" "clock" "cross" "dotbox" "exchange" "fleur" "heart" "man" "mouse" "pirate" "plus" "shuttle" "sizing" "spider" "spraycan" "star" "trek" "watch"
Distance: (1) pixels ➔ numeric**;** (2) absolute distances ➔ strings with a trailing character denoting units: c-centimeters, i-inches, m-millimeters, p-printer's points - these vary by font
Fonts: A list or tuple specifying name, size, weight. Ex: font= ("Verdana", 10, "bold"). Font sizes with positive numbers measured in **points**; negative numbered sizes are measured in **pixels**.
Images: **B&W** id constructor: myBWpic = tk.BitmapImage (file=myimagefile.xbm); **Color** myphotoimage = PhotoImage (file=myimagefile. {.gif, .pgm, .ppm formats}) see pg 7
Justify: "left", "center", "right", "fill" ➔ *include the quotes*
Region: 4 space-delimited distances **"3i 2i 4.5i 2i"**
Relief: "raised", "sunken", "flat", "groove", "ridge"

Wrap: "none", "char", or "word" - for example: text box attribute

WIDGETS

Button	Checkbutton	LabelFrame	Message	Scrollbar
Canvas	Entry	Listbox	PanedWindow	Spinbox
	Frame	Menu	Radiobutton	Text
	Label	Menubutton	Scale	Toplevel

Creating WIDGETS

Create a widget by naming it and setting it equal to a type (see above) then setting it's attributes and options. The first attribute set must be it's parent window. Options can also be set using the configure method, i.e., widg.config(bg="snow"). A primary callback is bound with **command=**function/method/lambda

Note: in this document the symbol ↳ means "returns" or "yields".

A Basic "Boilerplate" Header for Small Apps

```
#  by John Oakey for www.wikipython.com
#  get the module, like any other Python module
from tkinter import *
#  establish root (set variables if needed)
root = Tk()
root.attributes('-fullscreen', True) #set full screen
root.configure(background='SteelBlue4')
# configure root options
root.attributes("-alpha", 0.5)  #  and attributes (opacity in this case)

#  get info on screen size: showing the math here
scrW, scrH = root.winfo_screenwidth(),\
      root.winfo_screenheight()
#  build a string variable telling tkinter how to
#  create and position our Toplevel window
workwindow = str(1024) + "x" + str(768)+ "+"
+str(int((scrW-1024)/2)) + "+" +str(int((scrH-
768)/2))
#  create the Toplevel under the root parent, set its geometry and title
top1 = Toplevel(root, bg="light blue")
top1.geometry(workwindow)
top1.title("Top 1 - Workwindow")

#  prepare for slightly more complex apps
top1.attributes("-topmost", 1)
# make sure top1 is on top to start
root.update()  # but don't leave it locked in place
top1.attributes("-topmost", 0)  # in case you need to use 'lower' or 'lift'

#exit button - note: this example uses grid geometry manager
b0=Button(root, text="Egress",\
command=root.destroy)
b0.grid(row=0,column=0,ipadx=10, ipady=10,
pady=5, padx=5, sticky = W+N)

#  create any functions and global variables that widgets and processes will need
my_text = StringVar()
my_text.set("Hello World!")

#  ____________________
#  create the rest of your program here
l1=Label(top1, bg="snow", text=my_text.get(),
font=('Times New Roman',14,'bold'))
l1.grid(row=1,column=1,ipadx=20, ipady=20,
pady=50, padx=50)
#  ____________________
#  and finally, tell tkinter it may begin execution
root.mainloop()  # but stop right here
```

Inherent **class** bindings are defined for all widgets giving them default callback behaviors to certain events—when a bound event occurs, bindings execute your **method, lambda expression**, or **function** specified at the widget's creation, or "secondarily" added later with the **.bind** method.

A widget creation and binding syntax diagram: *b1 will have a linen background color and read "Push!", if clicked it runs the callback*

Your widget name options (ex: background color) function to bind when an **auto event** occurs

b1=**Button(parent**, bg= 'linen', text= 'Push!', **command=** my_callback_function)

widget type **parent window name** options (ex: label text) **button click binding is automatic*

© 2021 John A. Oakey V

Binding Responses When an Event (an activity or change in state) Effects an Object

Mouse button or **Keyboard** are by far the most common types of events (see right) but other types include **crossing, focus, exposure, configuration, colormap and:**
class: .bind_class()
ex: self.bind_class(w_type, "<event>", function) and
application: .bind_all('<some event>', function) ex: self. bind_all('<Key-Print>', self.__printScreen)

Names given to **event sequences** are strings of one one or more event patterns. In the widget definition (see page 1), "command =" sets the callback response.

Events (like a button click) can bind at 4 levels: Class, Application, Toplevel, and Instance (the most common). Many widgets have predefined or "inherent" events which will automatically call any "**command=**" statement in their definition.
No event information is passed back for inherent callbacks.

" + angle bracket enclosure: like **"<Alt Button 1>"**

"< **[opt modifier(s)] type [opt detail]** >"

Alt, Control, Double, Lock, Shift, Triple — like Enter or Return — like 1, 2, 3

"< Alt Button 1 >"

Secondary Bindings: Bindings, or bindings where event environmental objects must pass back, can be set with the **.bind** method: **widget.bind ("<event>", *callback*, add=None/"+")** This method connects the existing widget to an event, a callback action, and specfies if this binding supersedes others or is executed in addition to others. **ex:** l2.**bind**("<Button-1>", my_callback_func, add='+')The '+' allows multiple callbacks to execute. This type of binding also will "pass back" environmental and activity information (see left insert). An additional binding can use an existing callback by defining the function with *args in the parens: def callback(event,*args)

Information passed back from a **.bind** method callback event: **widget:** tkinter instance; **num:** button number; **x,y:** mouse position; **x_root, y_root:** mouse pos relative to toplevel; **char:** character code (as a str); **keysym:** key symbol; **keycode:** key code; **other: width, height** - new widget size (pixels); **type:** event type;

Common Events

Mouse Button events:
always in quotes like "<1>"
<Button-1> or **<1>**
<Button-2> or **<2>**
<Button-3> or **<3>**
<ButtonRelease-1>
<B1-Motion>
moving with <1> held down
<DoubleButton-1>
<Enter> mouse enters widget

Keyboard events:
<FocusIn> or **<FoucsOut>**
keyboard focus in/out of widget
<Return> "**Enter**" key
<*key*> : w.bind("<*key*>", callback) any keypress; or frame. Bind ("x", **callback**) where "x" is the x key
Note* *You may need to use w.focus_set() to activate.*

Special key bindings *each encased in "< >"; ex: "<Tab>";* Cancel *(the Break key)*, BackSpace, Tab, 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, and Scroll_Lock

tkinter Geometries - How Objects are Placed

ActiveState © owns and maintains tkinter

Suggested reading: https://www.**activestate.com**/resources/quick-reads/how-to-position-widgets-in-tkinter/

Once a widget (or any object) is created, tkinter uses one of three **geometries** to arrange, register and place them on a screen. Never mix these systems in the same master window! Each geometry has a number of **options** - see list on page 3.

pack: very simple to learn and use—ideal for a new user to experiment with widgets and attributes but not practical for most applications because it decides on all exact placements with the programmer specifying only the relative positions.

grid: the generally most useful of the geometries, grid assumes placement of widgets is done in columns and rows and so allows neat, though restrictive, layouts to be composed fairly easily. (It makes Microsoft Excel an excellent layout planning tool.) Grid automatically sizes the grid based on the size attributes of the widgets to be positioned.

place: allows precise absolute placement using x,y coordinates or sizing relative to another widget.

Using lambda to pass simple values in a binding
Ordinarily no variables are passed in an automatic binding and only event data in manual bindings. A lambda statement can be used to overcome this limitation with simple data - for example regular or control variables.
Auto binding examples:
command= lambda: button1callback(myvar) -or-
command= lambda: button1callback(**cvar.get()**)
the callback function must specify a receiving variable container. For a secondary bind: w.bind("<event>", lambda event, arg=variable: callback(event, arg))
callback has 2 variables such as (event, myvar)

Methods: Universal Geometry Methods

[x=widget. Geometry_method()] ex: b1.pack_forget()
x_forget() remove from manager but do not destroy, can reuse; ex: **lab1.grid_forget()**, retrieve by repeating the original grid command
x_info() ↳ a dictionary of options **w.grid_info()**
x_slaves() ↳ list of sub widgets as tkinter widget references. same as **x_content()**
x_configure(options) *same as .pack()*

Geometry Specific Methods

place: *has **no** other Methods.*

pack and grid:
x_propagate(window_name, T/F) ; 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
master.grid_location(x,y) ↳ r/c tuple with indexes
w.grid_remove() removes widget from manager; but the widget is 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=, uniform=

comments and suggestions appreciated:
oakey.john@yahoo.com

© 2021 John A. Oakey

Geometries: Pack, Place, Grid: widget placement & formatting commands Options and Attributes

Geometry Options	Pack	Grid	Place	Default or Requirement	Options	Note
after	●			other window		name of another window
anchor	●		●	center	compass points	default is NW
before *(other)*	●			other window		name of another window
bordermode			●	inside	outside / ignore	border influence on slave placement
column		●		0	integer	column number; columns start with 0
columnspan		●		1	integer	number of columns spanned
expand	●			FALSE	true/false: 0/1	assign more space, distribute among widgets
fill	●			"none"	"x", "y", "both"	take up entire space, may need expand=
height			●	size	distance	outer dimension of window plus border
in_= *(target)*	●	●	●	n/a	widget name	pack inside the target widget
ipadx	●	●		0	distance	internal padding pixels/distance; horizontal
ipady	●	●		0	distance	internal padding pixels/distance; vertical
padx	●	●		0	distance	external padding pixels/distance; horizontal
pady	●	●		0	distance	external padding pixels/distance; vertical
relheight			●	fp	fp 0 to 1.0	height relative to master; modified by - height
relwidth			●	fp	fp 0 to 1.0	width relative to master; modified by - width
relx			●	fp % * 100	0.0(left)-1.0(right)	anchor point x coordinates in master window
rely			●	fp % * 100	0.0(left)-1.0(right)	anchor point y coordinates in master window
row		●		first empty	row number	row number; rows start with 0
rowspan		●		1	integer	number of rows spanned
side	●			top /	left/ right/ bottom	to pack against
sticky (glue widget to cell border)		●		centered	compass points	W+E stretch horiz; W+E+N+S - fill all; use a **string="wens"** or **constants** =W+E+N+S
width			●	size	distance	outer dimension of window **plus** border
x			●	0	distance	anchor point x coordinate in master window
y			●	0	distance	anchor point y coordinate in master window

Information Methods (winfo_x())

ex: print(top1.winfo_children()) ↳ list of child objects

atom(name (a string),displayof=0): ↳ unique int mapped to str
atomname(id, displayof=0): ↳ text name mapped to id
cells(): ↳ # of cells in colormap, a decimal string
children(): ↳ list of children in stacking order, not toplevels
class(): ↳ widget class
colormapfull(): ↳ 1, if full
containing(*window* rootx, rooty, displayof=0):widget @ this pos
depth(): ↳ bit depth, pixels at this
exists(): ↳ true if widget exists
fpixels() *window, number* : fp-# of screen pixels
geometry(: ↳ geometry sting in pixels
height(): ↳ widget height in pixels
id(): ↳ window identifier
interps(displayof=0 *window*): ↳ TCL interpreter mem list
ismapped(): ↳ boolean; check if window created
manager(): ↳ geo mgr: grid,pack,place, or class command if wid
name(): ↳ widget name
parent(): ↳ widget parents full name
pathname(displayof=0 *window*) : full name of id # window
pixels(): ↳ *window,number* : convert to pixels
pointerx(): ↳ x coord of pointer on the root
pointerxy(): ↳ xy coords of pointer on the root
pointery(): ↳ *window* y coord of pointer on the root
reqheight(): ↳ min size to display widget
reqwidth(): ↳ min size needed to display widget
rgb(color,): ↳ *color*: an RGB 3 tuple - 0 to 65535
rootx(): ↳ left edge x coordinate relative to screen
rooty(): ↳ left edge y coordinate relative to screen
screen(): ↳ screen name as dec int; ":0.0" in Windows
screencells(): ↳ # of color cells
screendepth(): ↳ deffault bit depth
screenheight(): ↳ height of widget screen
screenmmheight(): ↳ screen height in mm
screenmmwidth(): ↳ screen width in mm
screenvisual(): ↳ "psuedocolor" or "truecolor"
screenwidth(: ↳ width of screen in pixels
server(): ↳ widget screen xserver window info
toplevel(): ↳ widget root
viewable(): ↳ True if widget chain is mapped
visual(): ↳ directcolor, grayscale, pseudocolor, staticcolor, staticgray, or truecolor
visualid(): ↳ X identifier for visual this widget
visualsavailable() : list of all visuals available for widget screen
vrootheight(): ↳ hght of the virtual root window for widget
vrootwidth(): ↳ width of the virtual root window for widget
vrootx():↳x offset of virtual root rel to root window of wid screen
vrooty(): ↳ y offset of virtual root rel to root window of wid scrn
width(): ↳ widget width, pixels (update_idletasks)
x() and **y**(): ↳ upper corner coordinates

Constants

Boolean: 0/'no', 1/'yes' **Anchor points:** 'n', 'ne', 'e','se','s','sw','w','nw','center' **Bitmaps:** 'error','gray75', 'gray50', 'gray25', 'gray12', 'hourglass', 'info', 'questhead', 'question', 'warning'

© 2021 John A. Oakey

Widget Options and Attributes	Button	Canvas	Checkbutton	Entry	Frame	Label	LabelFrame	Listbox	Menu	Menubutton	Message	PanedWindow	Radiobutton	Scale	Scrollbar	Spinbox	Text	Toplevel	Value Type	Default Value	Note
activebackground	●		●			●			●	●			●	●	●	●			color	a system value	mouse over or b1 press
activeborderwidth									●										distance value	1	menu; only if displaying > 1 element
activeforeground	●		●			●			●	●			●						color	a system value	
activerelief															●				relief	raised	displayed when element active
activestyle								●											dotbox, none, underline	underline	
anchor	●		●			●				●	●		●						compass points, center	usually center	
aspect											●								positive integer aspect ratio	150	message;
autoseparators																	●		boolean	TRUE, 1	text; applies only if undo is true
background or -bg	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		color	system	
bigincrement														●					fp	0.1	large movement amoount
bitmap	●		●			●				●			●						"" or filename	see predefined list	see native available: "question"
blockcursor																	●		boolean	false	text;
borderwidth or -bd	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	distance value	2 pixels	
buttonbackground																●			color	SystemButtonFace	spinbox;
buttoncursor																●			cusor name	default cursor	spinbox;
buttondownrelief																●			relief	raised	spinbox;
buttonuprelief																●			relief	raised	spinbox;
class					●		●											●	class for window - determines options	w class name ->	toplevel, frame, labelframe
closeenough		●																	fp value	1.0	how close is inside
colormap					●		●											●	new' or other window	screen default	toplevel, frame, labelframe;
command	●		●										●	●	●	●			callback function name	none	
compound	●		●			●				●			●						bottom, top, left, right, center	none	controls the display of text and images at same time
confine		●																	boolean	True	canvas; T confines view to scroll region
container					●													●	boolean	false	toplevel, frame, labelframe; to embed app
cursor	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	cursor name	system	see cursor options
default	●																		normal, active, disabled	disabled	button; disabled may draw smaller button
digits														●					integer	0	scale; string resolution, 0 yeilds all positions
direction										●									above, below, left, right, flush	below	menubutton; where menu pops up
disabledbackground				●												●			color	system	spinbox, entry;
disabledforeground	●		●	●		●		●	●	●			●			●			color	SystemDisabledText	
elementborderwidth															●				pixels	=borderwidth?	around arrows and slider
endline																	●		next-after-last line index integer		text only;
exportselection				●				●								●	●		1 or 0	1	
font	●		●	●		●	●	●	●	●	●		●	●		●	●		font 3 tuple	TkDefaultFont	name, size, weight
foreground or -fg	●		●	●		●	●	●	●	●	●		●	●		●	●		color	system	
from														●		●			fp - lowest value	0.0	scale, spinbox; used with to and increment
handlepad												●							distance value	8	paned window; sash to handle distance
handlesize												●							distance value	8	paned window; side length of sash handle
height	●	●	●		●	●	●	●		●		●	●				●	●	text-lines, image-distance value	1	centered
highlightbackground	●	●	●	●	●	●	●	●		●	●		●	●	●	●	●	●	color	system	
highlightcolor	●	●	●	●	●	●	●	●		●	●		●	●	●	●	●	●	color	system	
highlightthickness	●	●	●	●	●	●	●	●		●	●		●	●	●	●	●	●	distance value	1 pixel	width of highlight if widget has input focus
image	●		●			●				●			●						bit flie: gif, pgm, ppm		image created with image_create command
inactiveselectbackground																	●		color		text; for selection when window does not have focus - no default
increment																●			fp(can be + or -)	1.0	spinbox; widget adj amount
indicatoron			●							●			●						boolean	false	check, radio, menu buttons; raised if true
insertbackground		●		●												●	●		color	black	
insertborderwidth		●		●												●	●		distance value	0	
insertofftime		●		●												●	●		milliseconds	300	
insertontime		●		●												●	●		milliseconds	600	
insertunfocussed																	●		none,(no cursor), hollow, solid	none	text; new in tcl8l.6
insertwidth		●		●												●	●		distance value	2 pixels	width of insertion cursor

BIG DADDY'S
© 2021 John A. Oakey
JR2021D

PYTHON TOOLBOX For 3.5

tkinter 0221 Toolbox Reference

Widget Options and Attributes	Button	Canvas	Checkbutton	Entry	Frame	Label	LabelFrame	Listbox	Menu	Menubutton	Message	PanedWindow	Radiobutton	Scale	Scrollbar	Spinbox	Text	Toplevel	Value Type	Default Value	Note
invalidcommand or -invcmd				●												●			script to evaluate	-	spinbox, entry; best use is "bell"
jump															●				boolean F-smooth T-update at release	0	drag slider value change amt
justify	●		●	●		●				●	●		●			●			see justify choices	center	
label														●					string	-	scale; label for the scale
labelanchor							●												compass points	"nw"	lableframe; text option can not be empty
labelwidget							●												widget	-	lableframe; widget to use as label
length														●					distance value	100	scale; long dimension of the scale
listvariable								●											name of a global variable		listbox; a list to be display in the widget
maxundo																	●		integer	0	text; 0 or neg yield unlimited undo stack
menu										●								●	associated menu name		toplevel, menubutton; **<-not identical**
offrelief			●										●						relief	raised	checkbutton, radiobutton; when button off
offvalue			●																value	0	checkbutton; variable when not selected
onvalue			●																value	1	checkbutton; variable when selected
opaqueresize												●							boolean	1	panedwindow; true-resize as sash moved
orient												●		●	●				'horizontal', 'vertical'	panwin is horiz	scale, scrollbar are vertical
overrelief	●		●										●						relief	"" (empty string)	button, checkbutton, radiobutton; exposed during mouse over
padx	●		●		●	●	●			●	●		●				●	●	distance value	1 pixel	
pady	●		●		●	●	●			●	●		●				●	●	distance value	1 pixel	
postcommand									●										a string holding ?		
proxybackground												●							color	background color	
proxyborderwidth												●							pixels	2	
proxyrelief												●							relief	flat	
readonlybackground				●												●			color	normal bg	spinbox, entry; if read only
relief	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	see relief choices	"sunken"	
repeatdelay	●													●	●	●			ms before engage	300	
repeatinterval	●													●	●	●			ms between execution	100	
resolution														●					real value	1 (integral)	scale; resolution of the scale
sashcursor												●							mouse cursor	hrzr: sb_h_ double_ arrow, vrt: sb-v-double_arrow	panedwindow; cursor when mouse over
sashpad												●							distance value	0	panedwindow; pad on each side of sash
sashrelief												●							relief	flat	panedwindow; sash relief
sashwidth												●							distance value	3	panedwindow; width of sash
screen																		●	screen name for new window		toplevel; can not modify with config
scrollregion		●																	coords: fp		rectangle: params or list
selectbackground		●		●				●								●	●		color	SystemHighlight	
selectborderwidth		●		●				●								●	●		distance value	0	
selectcolor			●						●				●						color	SystemWindow	checkbutton, radiobutton; use when selected
selectforeground		●		●				●								●	●		text color	SystemHighlightText	
selectimage			●										●						image	ignored unless image option specified	checkbutton, radiobutton; when button selected
selectmode								●											single, browse, multiple, extended	browse	listbox; styles for manipulatin the selection
setgrid								●									●		boolean	0	listbox, text; widget controls toplevel grid size
show				●															string - 1 character	-	entry; replaces entry - like "*" for password
showhandle												●							boolean	0	panedwindow; sash handles shown or not
showvalue														●					boolean	1	scale; show current value
sliderlength														●					distance value	30	scale; sze of slider
sliderrelief														●					relief	raised	scale;
spacing1																	●		distance value	0	text; + space above text lines

© 2021 John A. Oakey
JR2021D

PYTHON tkinter 0221 Toolbox Reference TOOLBOX For 3.5

Widget Options and Attributes	Button	Canvas	Checkbutton	Entry	Frame	Label	LabelFrame	Listbox	Menu	Menubutton	Message	PanedWindow	Radiobutton	Scale	Scrollbar	Spinbox	Text	Toplevel	Value Type	Default Value	Note
spacing2																	●		distance value	0	text; + space above single wrap lines
spacing3																	●		distance value	0	text; + space above last wrap line
startline																	●		integer index or ""	-	text; indicate line to start, "" is first line
state	●	●	●	●		●		●		●			●	●		●	●		Normal or Disabled	normal	
tabs																	●		distances for stops	8 characters	stops can preceed left, right, center, numeric
tabstyle																	●		'tabular' or 'wordprocessor'	'tabular'	text;
takefocus	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	●	0 or 1	1	
tearoff									●										boolean	0	
tearoffcommand									●										compound string	->	https://www.tcl.tk/man/tcl8.6/TkCmd/menu.htm#M8
text	●		●	●		●	●			●	●		●						a string	-	
textvariable	●		●	●		●				●	●		●			●			a string	-	
tickinterval														●					real number	0.0	scale; tick spacing - if 0 no tick marks
title									●											-	
to														●		●			real number	scale 100, spin 0.0	scale, entry; right or bottom end
tristateimage			●										●						image	-	ignored if image unspecified
trisatevalue			●																a string holding ?	-	
troughcolor														●	●				color (not Windows)	SystemScrollbar	scale, scrollbar;
type									●										menu entry type	normal	command, separator, tearoff
underline	●		●			●				●			●						integer	-1	index of char to underline
undo																	●		boolean	1	text; undo mechanism is active or not
use																		●	hex string window identifier	-	toplevel; get from winfo id
validate				●												●			none, focus, focusin, focusout, key, or all	none	spinbox, entry; specifies validation mode
validatecommand or -vcmd				●												●			script to eval, {} disables	{}	spinbox, entry; must return boolean value
value													●						proper list	-	radiobutton; content control w/ precedence over from to
values																●			proper list	-	spinbox; content control
variable			●										●	●					global variable	SelectedButton value	checkbox, radiobutton, scale;
visual					●		●											●	see TK_GetVisual for info/complex	parent values	toplevel, frame, lableframe;
width	21	●	0	20	0	0	0	20		0	0	''	0	15	21	20	80	0	characters	varies by widget	canvas=473
wrap																●	●		string - char or word	char	spinbox, text; wrap around values of data
wraplength	●		●			●				●			●						distance value	nowrap / 0	max line length; 0-none
xscrollcommand		●		●				●								●	●		=scrollbar.set		scrollbar.config(command=w.xview)
xscrollincrement		●																	distance	0	canvas; increment or horiz scroll
yscrollcommand		●						●									●		widget name + set		canvas, text, listbox;
yscrollincrement		●																	distance	0	canvas; increment or vert scroll

tkinter Commands and Keywords

The following are commands OTHER than those which create and define widgets, implement attributes and options, bind actions, provide information and manage window activity. (**Commands** is the web folder name used by the TCL/TK manual pages for these.) In the following, {Tk} or {Tcl} notes mean "see that specific command page" on the Tcl/Tk website.

after {Tcl} (1) **after(delay in milliseconds)** *nothing executes during the delay (2) **after(delay in milli-seconds, command to evaluate)** ex: e1.after(1000, top1.bell()) <- wait a second then play a tone

bell() {Tk} - causes a sound to play: ex: widget.bell()

bind {Tk} **–** see pages 1 and 2

bindtags {Tk} – produces a list of associations (a window/ widget, class name, or "all" keyword) called binding tags that determines how events are processed. Ex: bindlist= [widget.bindtags()] might yield : [('.!toplevel.!mylabel', 'Label', '.!toplevel', 'all')]

bitmap {Tk} – Do not confuse the 10 "built-in" bitmaps that can be called by name for place-ment on buttons (see page 3 for a list with pictures), with other (i.e., user defined) bitmap images. **Ex: b1=Button(top1, bitmap= 'questhead')** will yield, in your GUI, something like at right:

BitmapImage Class: see "Working with Graphics" on p.7

clipboard {Tk} – tkinter has its own clipboard. Multiple comments on the web give conflicting reports about its efficacy exchanging data with various operating systems, but the Tk/Tcl docs do not make the claim that it does that. There are three clipboard commands for **clear()**, **append()** and **get()**. You must clear it before calling append. Examples:

```
top1.clipboard_clear() #test clipboard -start by clearing it
top1.clipboard_append("e1 is now: " + e1.get())  # put string data on clipboard from an entry widget named "e1"
l4.configure(text=top1.clipboard_get())   retrieve the clipboard
```

© 2021 John A. Oakey V

Commands-Keywords continued

colors {Tk} – you can find a lot about colors on www.wikipyton.com. You can also find an "official" list of shortcut color names on the Tcl/Tk web site at: https://www.tcl.tk/man/tcl8.6/TkCmd/colors.htm

console {Tk} - Beyond this doc's scope, see: https://www.tcl.tk/man/tcl8.6/TkCmd/console.htm -

cursors {Tk} -define inside configure, i.e., e1.config (cursor="plus"); example: "coffee_mug"; see whole list at https://www.tcl.tk/man/tcl8.6/TkCmd/cursors.htm

destroy {Tk} – deletes window and all children example: root.destroy() – if all destroyed, normal exit occurs.

event {Tk} – a series of commands to manage virtual and synthesized events

w.event_add("<< your new virtual event>>", sequences)

w.event_delete("<< your virtual event>>", sequences)

more: https://www.tcl.tk/man/tcl8.6/TkCmd/event.htm

focus {Tk} – 5 Python variations

w.focus_set() – moves focus to widget w

w.focus_displayof() – ↳ w with focus if it is on current display, else "None"

w.focus_force() – override window manager and force focus to w, not considered good programming: "impolite"

w.focus_get() – if app is active, ↳ w with focus, else 'None'

w.focus_lastfor() – ↳ the widget that last had the input focus in the top-level window that contains it

font {Tk} – in modern Python it is no longer necessary to load a font submodule (fkFont) to change major font characterists; 12 of 18 widgets allow setting the font it uses with a 1 to 6 item tuple in a string; usually with spaces for delimiters, for example, w.configure(font= ("Arial 8 bold italic underline overstrike")). If we want a font name with spaces in names, a different format/ syntax is necessary, for example: w.configure(font= ("Courier New", 12, "bold")). Note that if you drastically increase the font size of a widget that already holds text, the widget size may increase to fit the text displayed.

grab {Tk} – **w.grab_set()** for example: self.top.grab_set() – where top is a toplevel window the grab command limits all activity to that window.

Variations:

w.grab_current() - if grab in force ↳ identifier, else 'None'

w.grab_release() – end any grab in force

w.grab_set_global() – considered bad if not evil programming, grabs all events for everything!

w.grab_status() – ↳ 'local' or 'global' or 'None'

grid {Tk} – along with **pack** and **place** these are the geometry managers and are just commands like the rest of this list.

image {Tk} – see "Working with Graphics"

keysyms {Tk} – TK recognized list of names recognized for bindings (ex: 'space', 'exclaim', 'quotedbl', 'percent') can be found at: https://www.tcl.tk/man/tcl8.6/TkCmd/keysyms.htm

lift - in Python implementations, **lift(aboveThis=None) replaces** Tk's "raise(aboveThis=none)" because "raise" is a keyword in Python.

loadTk {Tk} – "loads binary code from a file into the application's address space and calls an initialization procedure in the package to incorporate it into an interpreter" - beyond the scope of this doc or the knowledge of its author.

lower(belowThis=None) {Tk} – change a window's position in the stack. "All toplevel windows may be restacked with respect to each other, whatever their relative path names, ***but the window manager is not obligated to strictly honor requests to restack***."

option {Tk} – there are several variations of the option command – one of which is dubious **w.option_add(pattern,value,priority)** explaination and examples by Fredrik Lundh himself in 1998, one of his examples: root.option_add("*Font", "courier"). The pattern format is explained at https://www.tcl.tk/man/tcl8.6/TkCmd/option.htm#M9 and priority values are explained by Shipman at https://anzeljg.github.io/rin2/book2/2405/docs/tkinter/index.html; briefly, priorities (default=80) are: 20 – global, 40 – applications, 60 – user files, 80-option set up after app initialization.

w.option_clear() – self descriptive

w.option_readfile(filename, priority**) –** *not tested*

w.opton_get(name-instance key, className – class key) – *This command appears to have a bug:* Since all widgets support cget(), it should be superfluous anyway. options {Tk} – see standard options list: https://www.tcl.tk/man/tcl8.6/TkCmd/options.htm

PhotoImage class {Tk} See "Working with Graphics"

raise {Tk} – RAISE DOES NOT EXIST in the Python implementation; the substituted command is toplevel_obj.**lift(aboveThis=None)** – also see **lower()**

selection {Tk} – implements the full selection functionality described in the X Inter-Client Communication Conventions Manual (ICCCM). This includes **selection_clear(), _get(), _handle(command), _own(), _own_get()**.

send {Tk} – execute a command in a different application see https://www.tcl.tk/man/tcl8.6/TkCmd/send.htm

update() – force process of all pending events, can be used to force response to user inputs.

update_idletasks() – processes only display and window calculations – does not allow new events

Working with Graphics

For working with the vast majority of graphic types, the coder will largely need to use the **Pillow** Module—details of which are left to the research of the reader. In modern Python **Pillow** replaces the old Python Image Library (PIL).

tkinter natively handles two types of images: (bi-color and full color) which can be displayed in four widgets: text, button, label and canvas.

Canvas widgets are dedicated to graphic images, lines, shapes and text and are outside the scope of this toolbox. Note that the canvas **create image()** method requires an image first established by the PhotoImage process below. Bitmap and photo images are created from a file and assigned a variable name which can then be used anywhere.

(1) BitmapImage - monochrome X11 files with the exten -sion **".xbm"**. Normally black on transparent, the syntax: **name = BitmapImage(file =r"path and file name" [, bg=color][, fg=color])** allows background and foreground to be set to colors. Resizing or converting xbm images must be done outside tkinter.

(2) PhotoImage - full color **.gif**, **.pgm** (gray scale 0 to 65536), **.ppm** and **.png [**added in 8.6**]** images are created with **name=PhotoImage(file=r"path and file name")** Once a name is created it is used to set its display in a widget. **Ex:** (assume pic to be in Label lb0 which is in top1)

img_obj = **PhotoImage**(file = r"path_and_file")
lb0 = Label(top1, image=**img_obj**).grid(row=3, column=3)

It is good practice to keep a reference to your image stored in a global variable.

Methods for PhotoImage objects include:

* We did not test all of the methods listed here.

blank() - display a transparent image
cget(option) - return the value of option
configure()
copy() - return a new photoimage with the same image
get(x,y) - return the value of the pixel at x.y
height()
put(data, to=None)
put(data,bbox)
read() and **write()**
subsample (scale) return new image using every xth
subsample(xscale, yscale) pixel, y=x if not provided
Ex: given a created image object, imgobj, we can reduce its size by 75% with : **imgobj = imgobj.susample(4)**
type()
width()
zoom(scale) - return new image zoomed by x,y
zoom(xscale,yscale)

Compliments of: BIG DADDY & www.wikipython.com

© 2021 John A. Oakey V

Widget Methods and Options

Two important **tkinter.Misc** methods are **universal**:
cget (get an option value)
configure (set an option value)
} Six have **NO** widget-specific commands: **Frame, Label, LabelFrame, Message, Menubutton, Toplevel** Most other **widget-specific** methods are listed below:

***self is usually first argument** - not shown here **tagOrId is a search-spec compliant item*

Button

flash()
invoke (*existing callback*)

CheckButton

flash()
invoke() *toggles select and call an existing callback*
select() set on state
deselect() set off state
toggle() toggle and invoke a command if given

Canvas

EX: canvasobj.addtag_above(newtag, tagOrId)

addtag(*args; *internal func*)
_above(newtag, tagOrid)
_all all item in the canvas
_below item below tagOrId
_closest(newtag, x, y, halo, start)
*_enclosed(*newtag,x1,y1,x2,y2)
*_overlapping(*newtag,x,y,x2,y2)
_withtag(newtag, tagOrId)
bbox(*args) ↳ a tuple of X1,Y1,X2,Y2 for args
canvasx(screenx, gridspacing=None)
canvasy(screeny, gridspacing=None)
coords(*args) list of coords
create create_arc()
_arc(*args, **kw) arc w. x1,y1,x2,y2
_bitmap(*args, **kw) coord x1,y1
_image(*args, **kw) coord x1,y1
_line(*args, **kw) coord x1,y1,...,xn,yn
_oval(*args, **kw) coord x1,y1,x2,y2
_polygon(*args, **kw) coord x1,y1,...,xn,yn
_rectangle(*args, **kw) coord x1,y1,x2,y2
_text(*args, **kw) coordinates x1,y1
_window(*args, **kw) coord x1,y1,x2,y2
dchars(*args) delete chars of text items tag or id in ARGS from FIRST to LAST
delete(*args) items identified by all tag or ids
dtag(*args) delete tag or id given as last arguments in ARGS from items identified by 1st argument in ARGS
find(*args) Internal funct
_above(tagOrId)
_all()
_below(tagOrId)
_closest (x, y, halo=None, start=None)
_enclosed (x1,y1,x2,y2)
_overlapping(x1,y1,x2,y2)
_withtag(tagOrId)
focus(*args) first item spec
gettags(*args)1st itemspec
icursor(*args) set at POS in TAGOROD
index(*args) cur pos(int)
insert(*args)TEXT in item TAGORID at position POS ARGS must be TAGORID POS TEXT.
itemcget(tagOrId, option) ↳ value for OPTION for item TAGORID
itemconfig =
itemconfigure(tagOrId, cnf=None, **kw) resources item TAGORID
lift =
tag_raise(*args)
lower =
tag_lower(*args)
move(*args) TAGORID given in ARGS
moveto(tagOrId, x='', y='') *see docs*
postscript(cnf={}, **kw) see docs
scale(*args) item TAGORID with XORIGIN, YORIGIN, XSCALE, YSCALE.
scan_dragto(x, y, gain=10) Adjust canvas to GAIN times the diff between X and Y and the coord given in scan_mark
scan_mark(x, y) Note X, Y coord
select_adjust(tagOrId, index) adjust selection TAGORID to index
select_clear() clear the selection
select_from(tagOrId, index) set fixed end of TAGORID to INDEX
select_item() ↳ selected item
select_to(tagOrId, index) Set variable end of TAGORID to INDEX
tag_bind(tagOrId, sequence =None, func=None, add= None)
Bind to all items with TAGORID at event SEQUENCE a call to FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind ↳ value.
tag_lower(*args) lower an item TAGORID
tag_raise(*args) raise an item TAGORID
tag_unbind(tagOrId, sequence, funcid=None)
tkraise = tag_raise(*args)
type(tagOrId) ↳ TAGORID type

Entry

delete(first, last=None)
get() ↳ the text
icursor(index) Insert at INDEX.
index(index) pos of cursor
insert(index, string)
scan_dragto(x) Adjust the view of the canvas to 10 times the difference between X and **Y** and the coords given in scan_mark.
scan_mark(x) note cur X, Y coordinates
selection
_adjust(index)
_clear()
_from (index)
_present
_range(start, end)
_to(index)

Listbox

activate(index) item identified by INDEX
bbox(index) ↳ tuple of X1,Y1,X2,Y2 for the given index
curselection() indices of currently selected item
delete(first, last=None) last inclusive
get(first, last=None) list of items
index(index) item INDEX
insert(index, *elements) Insert ELEMENTS at INDEX
itemcget(index, option) value for ITEM OPTION
itemconfig =
itemconfigure(index, cnf=None, **kw) Configure resources of an ITEM. The values for resources are background, bg, foreground, fg, selectbackground, selectforeground
nearest(y) get item index nearest to y coordinate Y.
scan_dragto(x, y) Adjust the view of the listbox to 10 times the difference between X and Y and the coordinates given in scan_mark.
scan_mark(x, y) Note current X, Y coords
see(index) Scroll such that INDEX is visible.
select_anchor=
select_clear =
select_includes =
select_set =
selection_anchor(index) Set the fixed end of the selection to INDEX.
selection_clear(first, last=None) last included
selection_includes(index) True if selected
selection_set(first, last= None) current unchanged
size() ↳ the number of elements in the listbox.

Menu

activate(index) entry at INDEX
add(itemType, cnf={}, **kw) Internal function
_cascade(cnf={}, **kw) hierarchical menu item
_checkbutton(cnf={}, **kw) checkbutton item
_command(cnf={}, **kw) command item
_radiobutton(cnf={}, **kw) radio menu item
_separator(cnf={}, **kw) separator
delete(index1, index2= None) menu items between INDEX1 and INDEX2
entrycget(index, option) ↳ resource value of a item for OPTION at INDEX
entryconfig=
entryconfigure(index, cnf= None, **kw) Config menu item at INDEX
index(index) ↳ the index of menu item INDEX
insert(index, itemType, cnf= {},**kw) Internal function.
_cascade(index, cnf={}, **kw) Add hierarchi-cal menu item at INDEX
_checkbutton(index, cnf= {}, **kw) Add check-button item at INDEX
_command(index, cnf= {}, **kw) Add command menu item at INDEX
_radiobutton(index, cnf= {}, **kw) Add radio menu item at INDEX
_separator(index, cnf= {}, **kw) Add sep at INDEX.
invoke(index) menu item at INDEX and execute the associated command.
post(x, y) Display a menu at position X,Y.
tk_popup(x, y, entry='') Post the menu at position X,Y with entry ENTRY.
type(index) ↳ the type of the menu item at INDEX.
unpost() Unmap a menu.
xposition(index) ↳ the x-pos of the leftmost pixel of the menu item at INDEX
yposition(index) ↳ the y-pos of the topmost pixel of the menu item at INDEX

© 2021 John A. Oakey V

Widget Methods\Options - 2

Menubutton

obsolete since Tk8.0

PanedWindow

add(child, **kw) add a child widget followed by pairs of options and values accepted by the paneconfigure method: handlepad, handlesize, opaqueresize, sashcursor, sashpad, sashrelief, sashwidth, showhandle
forget = **remove**(child)
panecget(child, option) Query a management option for window.
paneconfig = **paneconfigure**(tagOrId, cnf=None, **kw) query or modify opts for window:
after window
before window
height size (TK_GetPixels value)
minsize n
padx n (TK_GetPixels value, + only)
pady n (TK_GetPixels value, + only)
sticky style a string of 'n', 's', 'e' or 'w'. If a pane is larger than the window, this will position or stretch the window within its pane. If opposites are specified, the window is stretched to fill the entire height (or width) of its cavity.
width size outer dimensions including border (TK_GetPixels value)
panes() an ordered list of the child panes.
proxy(*args) Internal function.
proxy_coord() x and y of most recent proxy location
proxy_forget() remove from display
proxy_place(x, y) at the given x and y coordinates
remove(child) remove the pane containing child from the panedwindow
sash(*args) Internal function
sash_coord(index) ↳ current x and y pair for the sash given by index, an integer between 0 and 1 less than the number of panes in the panedwindow
sash_mark(index) Records x and y for the sash given by index; used with dragto commands
sash_place(index, x, y) place at the given cords

RadioButton

deselect() put in off-state
flash() flash button
invoke() Toggle the button and invoke a command if given as resource.
select() put in on-state

Scale

coords(value=None) Return a tuple (X,Y) of the point along the centerline of the trough that corresponds to VALUE or current value if None is given.
get() value as int or float
identify(x, y) ↳ where the point X,Y lies. Valid return values are "slider", "though1" and "though2".
set(value) Set the value to VALUE

Scrollbar - *create and attach to a parent widget*

activate(index=); *sel elem "arrow1", "slider", "arrow2"*
delta(Δx Δy) *fraction chg*
fraction(*x,y*) ↳ *fraction of slider, i.e. position*
get()*cur fractions of slider*
identify(x,y) ↳ *element under x,y*
set(first%, last%) *visible*

Spinbox

bbox(index) ↳ coord tuple enclosing char index: x,y (upper left),width, height (pixels)
delete(first, last=None) one or more elements, ↳ an empty string
get() ↳ the spinbox's string
icursor(index) alter cursor insert position, ↳empty str
identify(x, y) name of the widget at x, y; ↳ is 'none', 'buttondown', 'buttonup', 'entry'
index(index) ↳ the numerical index of index
insert(index, s) Insert string at index. ↳ an empty string.
invoke(element) buttondown or buttonup, triggering associated action
scan(*args) Internal function
scan_dragto(x) Compute Δ between x and the x argument to the last scan mark command; adjust the view left or right by 10 times the Δ x-coordinates; high speed spinbox effect using mouse, ↳ "".
scan_mark(x) Records x and current view in the spinbox window; used in conjunction with later scan dragto commands Typically this command is associated with a mouse button; ↳ ""
selection(*args) Internal function
selection_adjust(index) find end of selection nearest index, adjust that end of the selection to be index
selection_clear() Clear the selection.
selection_element (element= None) set or get currently selected element
selection_from(index) Set the fixed end of a selection to INDEX.
selection_present() ↳ True if characters selected in the spinbox.
selection_range(start, end) Set the selection from START to END (not included).
selection_to(index) Set the variable end of a selection to INDEX.

Text

bbox(index) ↳ tuple x, y, width, height which gives the bounding box of the visible part of the char index.
compare(index1,op,index2) ↳ whether between index INDEX1 and index INDEX2 the relation OP is satisfied. OP is one of <, <=, ==, >=, >, or !=
count(index1,index2,*args) the number of relevant things between two indices. If index1 is after index2, result is neg number; Valid counting options are "chars", "displaychars", "displayindices", "displaylines", "indices", "lines", "xpixels" and "ypixels"
delete(index1, index2=None) between INDEX1 and INDEX2 (not included).
dlineinfo(index) ↳ tuple(x, y, width, height, baseline) giving the bound box and baseline pos of the visible part of the line containing the char at INDEX
dump(index1, index2= None, command=None, **kw) ↳ the contents of the widget between index1 and index2. The type of contents returned is filtered based on the keyword parameters;if 'all', 'image', 'mark', 'tag', 'text', or 'window' are given and true, then the corresponding items are returned. The result is a list of triples of the form (key, value, index). If none of the keywords are true then 'all' is used by default. If the 'command' argument is given, it is called once for each element of the list of triples, with the values of each triple serving as the arguments to the function. In this case the list is not returned.
edit(*args) Internal method The following forms of the command are currently supported: ex: edit_redo()
_modified(arg=None) Get or Set the modified flag. If arg is not specified, ↳ the modified flag of the widget.
_redo() Redo the last undone edit, Generates an error when the redo stack is empty.
_reset() Clears the undo and redo stacks
_separator() Inserts a separator (boundary) on the undo stack.
_undo() Undoes the last edit action. Generates an error when the undo stack is empty.
get(index1, index2=None) ↳ the text from INDEX1 to INDEX2 (not included).
image_cget(index, option) ↳ the value of OPTION of embedded image @ INDEX.
image_configure(index, cnf= None, **kw) Configure an embedded image at INDEX.
image_create(index, cnf= {}, **kw) Create an embedded image at INDEX.
image_names() ↳ all names of embedded images in this widget.
index(index) ↳ the index in the form line.char for INDEX.
insert(index, chars, *args) Insert CHARS before the characters at INDEX. An additional tag can be given in ARGS.
mark_gravity(markName, direction=None) Change the gravity of a mark MARKNAME to DIRECTION (LEFT or RIGHT). ↳ the current value if None is given for DIRECTION.
mark_names() ↳ all mark names.
mark_next(index) ↳ the name of the next mark after INDEX.
mark_previous(index) ↳ the name of the previous mark before INDEX.

© 2021 John A. Oakey V

Widget Methods and Options - cont 3

Text continued

mark_set(markName, index) Set mark MARKNAME before the character at INDEX
mark_unset (*markNames) Delete all marks in MARKNAMES
peer_create(newPath-Name, cnf={}, **kw) Creates a peer text widget with the given newPathName, and any optional standard configuration options
peer_names() ↳ a list of peers of this widget
replace(index1, index2, chars, *args) Replaces the range of characters be-tween index1 and index2 with the given characters and tags specified by args
scan_dragto(x, y) Adjust the view of the text to 10 times the difference bet-ween X and Y and the coords from scan_mark
scan_mark(x, y) Remember the current X, Y coords.
search(pattern, index, stopindex=None, forwards=None, backwards=None, exact=None, regexp= None, nocase=None, count=None, elide=None) ↳ the index of the first character of a match or an empty string.
see(index) Scroll such that the character at INDEX is visible
tag_add(tagName, index1, *args) Add tag TAGNAME to chars between INDEX1 and index2 in ARGS
tag_bind(tagName, seq-uence, func, add=None) Bind to all characters with TAGNAME at event SEQUENCE a call to function FUNC
tag_cget(tagName, option) ↳ the value of OPTION for tag TAGNAME.
tag_config = **tag_configure**(tagName, cnf=None, **kw) Configure a tag TAGNAME
tag_delete(*tagNames) Delete tags in TAGNAMES
tag_lower(tagName, belowThis=None) Change priority of tag TAGNAME such that it is lower than the priority of BELOWTHIS
tag_names(index=None) ↳ a list of all tag names
tag_nextrange(tagName, index1, index2=None) ↳ start and end index for the 1st sequence of characters between index1 & index2 which all have tag TAG-NAME. The text is search-ed forward from INDEX1.
tag_prevrange(tagName, index1, index2=None) ↳ a list of start and end index for the first sequence of chars between INDEX1 and INDEX2 which all have tag TAGNAME. The text is searched backwards from INDEX1.
tag_raise(tagName, aboveThis=None) Change priority of tag TAGNAME such that it is higher than the priority of ABOVETHIS.
tag_ranges(tagName) ↳ a list of ranges of text which have tag TAGNAME.
tag_remove(tagName, index1, index2=None) Remove TAGNAME from all characters between INDEX1 and INDEX2.
tag_unbind(tagName, sequence, funcid=None) Unbind for all characters with TAGNAME for event SEQUENCE the function identified with FUNCID.
window_cget(index, option) ↳ the value of OPTION of an embedded window at INDEX.
window_config = **window_configure**(index, cnf=None, **kw) Configure an embedded window at INDEX.
window_create(index, cnf={}, **kw) Create a window at INDEX.
window_names() ↳ all names of embedded windows in this widget

Compliments of: BIG DADDY & www.wikipython.com

Window Manager (wm) Methods

* self is always first argument – not show here for space **examples below assume "from tkinter import *" and "top1" is a Toplevel window on root=TK(); 'wm_' is a placeholder for a toplevel.() widget to which methods apply and stands for the name of a window + '.', for ex: wm_geometry might be **top1.geometry(w*h+x+y)**

aspect = wm_aspect (minNumer=None, minDenom=None, maxNumer=None, maxDenom=None) set ratio width/height between MINNUMER/MINDENOM and MAXNUMER/MAXDENOM. Return a tuple if no args
attributes = wm_attributes(*args)
(1) return platform values print (root.attributes())
(2) return specific opt value
(3) set value(s)
On Windows:
-alpha set translucent %, 0-100
-disabled
-fullscreen root.attributes('-fullscreen', True)
-toolwindow specifies style
-topmost specifies top or not, 0/1 top1.attributes("-topmost", 1)
-transparentcolor
On Mac unknown On Linux/Unix X11 none

client = wm_client(name=None) sets, stores or clears name in WM_CLIENT_MACHINE property
colormapwindows = wm_colormapwindows(*wlist) Put wlist into WM_COLORMAPWINDS. Return current list of WM_COLORMAPWINDS widgets if wlist is empty
command = wm_command (value=None) set or return in WM_COMMAND property the invoke function for the app
deiconify = wm_deiconify() on Windows raises widget and give it focus
focusmodel = wm_focusmodel (model=None) "active" means that this widget will claim focus itself, "passive" means that the window manage sets it
forget = wm_forget(window) window unmapped and no longer managed
frame = wm_frame() Return identifier for decorative frame of this widget if present
geometry wm_geometry newGeometry = None) Set geometry to NEWGEOMETRY of the form: **=width* height +x + y**. Return current value if None is given. top1.geometry(*workwindow*) where *workwindow* is a definition string
grid = wm_grid(, baseWidth=None, baseHeight=None, widthInc=None, heightInc=None) this widget will be resized on grid boundaries. WIDTHINC and HEIGHTINC are the width and height of a grid unit in pixels. BASE-WIDTH and BASEHEIGHT are the number of grid units requested in Tk_GeometryRequest.
Tk_GeometryRequest(*tkwin, reqWidth, reqHeight*)
group = wm_group(pathName=None) set or return group leader widget
iconbitmap = wm_iconbitmap(, bitmap=None, default=None) see Tk docs
iconify = wm_iconify() Display widget as an icon.
iconmask = wm_iconmask (bitmap=None) See https://www.tcl.tk/man/tcl8.6/TkCmd/wm.htm#M21
iconname = wm_iconname (newName=None) Set or return widiget icon name
iconphoto = wm_iconphoto (default=False,*args) set icon image(s)
iconposition = wm_iconposition (x=None, y=None)
iconwindow = wm_iconwindow (pathName=None) display path instead of icon
manage = wm_manage(widget) widget becomes stand-alone toplevel
maxsize = wm_maxsize(width=None, height=None) set or return max dimensions
minsize = wm_minsize(width=None, height=None) set or return min dimensions
(for maxsize and minsize if window is gridded values are grid units)
overrideredirect = wm_overrideredirect(boolean=None) Instruct the window manager to ignore this widget if BOOLEAN is given with 1. Return the current value if None given.

© 2021 John A. Oakey V

Window Manager Methods (wm) - continued

positionfrom = wm_positionfrom (who=None) see https://www.tcl.tk/man/tcl8.6/TkCmd/wm.htm#M21
protocol = wm_protocol (name=None, func=None) Bind function FUNC to command NAME for this widget.
resizable = wm_resizable (width=None, height=None) both values are boolean
sizefrom = wm_sizefrom (who=None) who="user" can change size
state = wm_state (newstate=None) Query or set the state of this widget as one of normal, icon, iconic (see wm_iconwindow), withdrawn, or zoomed (Windows only).
title = wm_title (string=None)
transient = wm_transient (master=None) with regard to Master
withdraw = wm_withdraw () unmapped, redraw widget with deiconify

Supporting tkinter Modules

These **modules** are part of tkinter but they must be imported - they do NOT import automatically with tkinter when it is imported.

Fonts in General: *Note that **fontchooser** is not in the **Python** module docs for a reason—it does not work.*

tkinter deals with fonts as **named** objects holding a dictionary of 5 arguments for each family: *family (name/string), size (int), weight ('bold'/'normal'), slant ('italic'/'roman'), underline (bool), and overstrike (bool).* tkinter maintains a small tuple of fonts that translate across systems and make it transportable. Those fonts can be seen with the **font.names()** method—see below.

FONT from tkinter import font

font.names() ↳ ('font1', 'fixed', 'oemfixed', 'TkDefaultFont', 'TkMenuFont', 'ansifixed', 'systemfixed', 'TkHeadingFont', 'device', 'TkTooltipFont', 'defaultgui', 'TkTextFont', 'ansi', 'TkCaptionFont', 'system', 'TkSmallCaptionFont', 'TkFixedFont', 'TkIconFont')
font.families() ↳ a tuple of all fonts on your system
font.nametofont("font family name"**)** creates a named font tuple that will provide tkinter with all the arguments necessary to utilize a font. ex: font.nametofont('ansifixed')
You can define your own font family with, for example:
fnew = (**font.Font**(family="Helvetica", size=24, weight='normal', slant='italic', underline=1))
A font object has these methods:
.actual(option=None, displayof=None) ↳ the attributes of the font.
.cget(option) Retrieve an attribute of the font.
.config(**options) Modify attributes of the font.
.copy() ↳ new instance of the current font.
.metrics() ↳ font attribute values. for example for myf1=font.nametofont('ansifixed'); myf1.metrics()
↳ {'ascent': 12, 'descent': 4, 'linespace': 16, 'fixed': 1}
.measure(text, displayof=None) space occupied by "text" in window specified in pixels.

SCROLLEDTEXT import tkinter.scrolledtext

Constructed like a Text widget but has a text widget and a vertical scroll bar packed in a frame. Special Attributes: ScrolledText.frame, and Scrolledtext.vbar

To use: (example assumes long text in variable "testtext" and gui set up - see standard header www.wikipython.com)

```
st= tkinter.scrolledtext.ScrolledText(top1, width=50,
wrap=WORD, height=8)
st.grid(row=1, column=1)
st.insert(INSERT, testtext)
```

Some stackoverflow opinions are that this widget is "buggy" and easier to construct from scratch

COLORCHOOSER from tkinter import colorchooser

```
rbgcolor=(); webcolor=""
rbgcolor, webcolor = colorchooser.askcolor(parent=top1, \
title="Select a Color", initialcolor="snow")
print(rbgcolor, webcolor)
#variables will hold something like:
```
↳ (255.99609375, 255.99609375, 128.5) #ffff80

COMMONDIALOG

provides the dialog **class** underlying other supporting modules including filedialog, colorchooser and messagebox (Lundh, 1997) **Helpful Definitions:**
Modal—blocks other actions until a window is dismissed
Nonmodal—remains active while other actions occur

TKINTER DIALOGS

SIMPLEDIALOG

from tkinter import simpledialog
simpledialog.**askfloat**(*title*, *prompt*, ***kw*)
simpledialog.**askinteger**(*title*, *prompt*, ***kw*)
simpledialog.**askstring**(*title*, *prompt*, ***kw*)¶
For example:
mynum = simpledialog.askfloat("Get a Float", "Input a float: ", parent=top1)

FILEDIALOG from tkinter import filedialog

Static functions:
*Options: parent, title, initialdir, initialfile, filetypes (a sequence of (label, pattern) tuples, * wildcard is OK*
Create Open dialog, return file in read-only mode:
filedialog.**askopenfile**(mode="r", **options)¶
filedialog.**askopenfiles**(mode="r", **options)
Create Save dialog, return file in write-only mode:
filedialog.**asksaveasfile**(mode="w", **options)
filedialog.**askopenfilename**(**options)
filedialog.**askopenfilenames**(**options)
Create a SaveAs dialog and return **selected** filename:
filedialog.**asksaveasfilename**(**options)
Prompt user **to select** a directory:
filedialog.**askdirectory**(**options)
Ex: newdir = filedialog.askdirectory(top1)
Build Your Own file/directory windows - see:
https://docs.python.org/3/library/dialog.html#module-tkinter.filedialog
filedialog.FileDialog(master, title)
cancel_command(event=None), dirs_double_event(event), dirs_select_event(event), files_double_event(event), files_select_event(event), filter_command(event=None), get_filter(), get_selection(), go(dir_or_file=os.curdir, pattern="*", default="", key=None), ok_event(event), quit (how=None), set_filter(dir, pat), set_selection(file)

MESSAGEBOX from tkinter import messagebox

Information: messagebox.**showinfo(*)**
Warnings: Messagebox.**showwarning (*)**
messagebox.**showerror (*)**
Questions:
messagebox.**askquestion (*)**
messagebox.**askokcancel (*)**
messagebox.**askretrycancel (*)**
messagebox.**askyesno (*)**
messagebox.**askyesnocancel (*)**

(*) ↳ (**title**=*None*, **message**=*None*,***options:* **default**= the default button like RETRY, ABORT, IGNORE; **parent** = *the window over which the messagebox is displayed*)

© 2021 John A. Oakey

tkinter 2021 Toolbox Reference — Section 2: tkinter *Ttk*

The addition of **Ttk** causes a major restructuring of tkinter. It replaces some familiar widgets, changing their functionality. It adds six new widgets. It reduces the options available for **all** widgets. It reduces the complexity of app design thru the use of **STYLES** and **THEMES**. The basic purpose of Ttk is to simplify code by separating methods and attributes and "automating" attributes with **styles,** emulating existing platforms, **or** providing other consistent appearances.

Replaced	Added
Button	Combobox
Checkbutton	Notebook
Entry	Progressbar
Frame	Separator
Label	Sizegrip
LabelFrame	Treeview
Menubutton	**Unaffected**
PanedWindow	Canvas
Radiobutton	Listbox
Scale	Menu
Scrollbar	Message
Spinbox	Text
** fg & bg are not supported by ttk	Toplevel

Importing/Setting up tkinter with Ttk

What follows is at odds with Shipman see: tkdocs.com/shipman/ but may be in line with https:// docs.python.org /3/library/tkinter.ttk.html#module-tkinter.ttk if it is interpreted literally. The following is the only process that worked for us across all demo tests.

```
import tkinter
from tkinter import *
from tkinter.ttk import *
ttk = tkinter.ttk
root = tkinter.Tk()
style = ttk.Style()
```

Every widget is made of a group of **elements** which usually have multiple value options. Every widget set has a **Style()** that defines it appearance. A widget's **style** specifies its elements, options and their arrangement. It's name is either the value of it's style configuration option, or the default for the widget defined by it's class. Usually the style is "T" + the class, (TButton). The exception: "Treeview", not "TTreeview".

To Configure a style: style.configure("myStyle Name", foreground= "red", cursor ="plus")

Themes are a **collection** of widgets of similar **styles**. Stock Ttk **themes** are: **winnative, clam, alt, default, classic, Vista,** and **xpnative.** After set up, to see available themes in your system from command prompt type: **style.theme_names()** Numerous **other themes** are available. Good resources: **https://tkdocs.com/tutorial/styles.html** and **https://wiki.tcl-lang.org/page/List+of+ttk+Themes**

To Use a Theme: style.theme_use("themename")

Widget States: 'active', 'disabled', 'focus', 'pressed', 'selected' ("on", "true", or "current"), 'background', 'readonly', 'alternate', 'invalid' (widget value not valid) - if prefaced by **!** the bit is OFF; i.e., the state is "**active**" or "**!active**"

Style: a class to manipulate the style database supporting the **style options** as follows:

configure(style, query_opt= None, **kw)
element_options (elementname)
element_names()
element_create (elementname, etype="image", or "from", *args, **kw= border, height, padding, sticky, and/or width)
map (style, query_opt=None, **kw)
lookup (style, option, state=None, default=None)
theme_create (themename, parent= None, settings=None)
theme_names()
theme_use (themename=None)
theme_settings (*themename*, settings)
layout (*style*, *layoutspec=None*) (None, side:whichside, sticky:nsew, unit: 0 or 1, children:[sublayout…]

Compliments of: BIG DADDY & www.wikipython.com

Ttk WIDGET ATTRIBUTES & OPTIONS	Button()	Canvas()	Checkbutton()	Combobox()	Entry()	Frame()	Label()	LabelFrame()	Listbox()	Menu()	Message()	Notebook()	PanedWindow()	Progressbar()	Radiobutton()	Separator()	Scale()	Scrollbar()	Sizegrip()	Spinbox()	Text()	Toplevel()	Treeview()
activebackground										☒													
activeborderwidth										☒													
activeforeground										☒													
activestyle									☒														
anchor							☒				☒												
aspect											☒												
autoseparators																					☒		
background		☒		☒	☒		☒		☒	☒	☒									☒	☒	☒	
bd		☒							☒	☒	☒										☒	☒	
bg		☒							☒	☒	☒										☒	☒	
blockcursor																					☒		
borderwidth		☒				☒	☒	☒	☒	☒	☒										☒	☒	
class	☒		☒	☒	☒	☒	☒	☒				☒	☒	☒	☒	☒	☒	☒		☒		☒	☒
closeenough		☒																					
colormap																						☒	
columns																							☒
command	☒		☒												☒		☒	☒		☒			
compound	☒		☒				☒								☒								
confine		☒																					
container																						☒	

© 2021 John A. Oakey

Ttk WIDGET ATTRIBUTES & OPTIONS	Button()	Canvas()	Checkbutton()	Combobox()	Entry()	Frame()	Label()	LabelFrame()	Listbox()	Menu()	Message()	Notebook()	PanedWindow()	Progressbar()	Radiobutton()	Separator()	Scale()	Scrollbar()	Sizegrip()	Spinbox()	Text()	Toplevel()	Treeview()
cursor	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒		☒	☒	☒	☒
default	☒																						
disabledforeground									☒	☒													
displaycolumns																							☒
end	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒		☒	☒	☒	☒
endline																					☒		
exportselection				☒	☒				☒											☒	☒		
fg									☒	☒	☒										☒		
font				☒	☒		☒		☒	☒	☒									☒	☒		
foreground				☒	☒		☒		☒	☒	☒									☒	☒		
format																				☒			
from																	☒			☒			
height		☒		☒		☒		☒	☒			☒	☒								☒	☒	☒
highlightbackground		☒							☒		☒										☒	☒	
highlightcolor		☒							☒		☒										☒	☒	
highlightthickness		☒							☒		☒										☒	☒	
image	☒		☒				☒								☒								
inactiveselectbackground																					☒		
increment																				☒			
insertbackground		☒																			☒		
insertborderwidth		☒																			☒		
insertofftime		☒																			☒		
insertontime		☒																			☒		
insertunfocussed																					☒		
insertwidth		☒																			☒		
invalidcommand				☒	☒															☒			
justify				☒	☒		☒		☒		☒									☒			
labelanchor								☒															
labelwidget								☒															
length														☒			☒						
listvariable									☒														
maximum														☒									
maxundo																					☒		
menu																						☒	
mode														☒									
offset		☒																					
offvalue			☒																				
onvalue			☒																				
orient													☒	☒		☒	☒	☒					
padding	☒		☒			☒	☒	☒				☒			☒								☒
padx											☒										☒	☒	
pady											☒										☒	☒	
phase														☒									
postcommand				☒						☒													
relief		☒				☒	☒	☒	☒	☒	☒										☒	☒	
screen																						☒	
scrollregion		☒																					
selectbackground		☒							☒												☒		
selectborderwidth		☒							☒												☒		
selectcolor										☒													
selectforeground		☒							☒												☒		
selectmode									☒														☒
setgrid									☒												☒		

© 2021 John A. Oakey
CC BY NC SA

Ttk WIDGET ATTRIBUTES & OPTIONS	Button()	Canvas()	Checkbutton()	Combobox()	Entry()	Frame()	Label()	LabelFrame()	Listbox()	Menu()	Message()	Notebook()	PanedWindow()	Progressbar()	Radiobutton()	Separator()	Scale()	Scrollbar()	Sizegrip()	Spinbox()	Text()	Toplevel()	Treeview()
show				☒	☒															☒			☒
spacing1																					☒		
spacing2																					☒		
spacing3																					☒		
startline																					☒		
state	☒	☒	☒	☒	☒		☒		☒						☒		☒			☒	☒		
style	☒		☒	☒	☒	☒	☒	☒				☒	☒	☒	☒	☒	☒	☒	☒	☒			☒
tabs																					☒		
tabstyle																					☒		
takefocus	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒
tearoff										☒													
tearoffcommand										☒													
text	☒		☒				☒	☒			☒				☒								
textvariable	☒		☒	☒	☒		☒				☒				☒					☒			
title										☒													
to																	☒			☒			
type										☒													
underline	☒		☒				☒	☒							☒								
undo																					☒		
use																						☒	
validate				☒	☒															☒			
validatecommand				☒	☒															☒			
value														☒	☒		☒						
values				☒																☒			
variable			☒											☒	☒		☒						
visual																						☒	
width	☒	☒	☒	☒	☒	☒	☒	☒	☒		☒	☒	☒		☒					☒	☒	☒	
wrap																				☒	☒		
wraplength							☒																
xscrollcommand		☒		☒	☒				☒											☒	☒		☒
xscrollincrement		☒																					
yscrollcommand		☒							☒												☒		☒
yscrollincrement		☒																					

Ttk Widget Methods and Options

Two important **tkinter.Misc** methods are **universal**: **cget** (get an option value), **configure** (set an option value); ttk.widget class also supports: **identify**(x,y) - name of element at that pos; **instate** (*statespec*, *callback=None*, **args*, ***kw)* True if state == statespec; **state** (statespec= None; "active", "disabled", "focused","pressed","selected","background","readonly","alternate","invalid","hover","normal", all negated by preface of "**!**"); **destroy**(self) Most other **widget-specific** methods are listed below (self not shown as the normal 1st attribute). Replaced widgets inherit most of their tkinter methods (not show here). For unaltered widgets see tkinter methods section, only a few new methods are added by Ttk.

Button
invoke() activate button command

Checkbutton
invoke() activate button command

Combobox
combines text field with a popdown list, inherits **Entry** methods and adds:
current(*newindex***)** sets value to element newindex or ↳ index (-1 if value not in list)
get() ↳ current value
set() sets current value

Entry
bbox(index**)** ↳ tuple of (x,y,width,height) which describes the bounding box of the character given by index
identify(x,y) name of element at x,y
validate() False if it fails

Frame
no new widgets

Label
no new widgets

LabelFrame
no new widgets

Listbox
no new widgets

Menubutton
obsolete since Tk8.0

OptionMenu
__getitem__(item) ↳ resource value for a key

Message
no new widgets

Notebook
TAB IDENTIFIERS(**tab_id**) int: 0 to # of tabs, child name, "@x,y", "current", "end" - ↳ # of tabs
add(child, **kw) adds a new tab
enable_traversal()
forget(tab_id**)**
hide(tab_id**)**
identify(x,y**)**
index(tab_id**)**
insert(pos, child, **kw**)**
select(tab_id=None**)**
tab(tab_id, option=None, **kw**)**
tabs() ↳ list of windows managed by notebook
[**Options:** see chart p.12]
[**tab options:** ***state*** (normal, disabled, hidden), ***sticky***, ***padding***, ***text*** (displayed in tab), ***image***, ***compound***, ***underline***]

PanedWindow
no new widgets
Progressbar
start(interval;default=50ms**)**
step(amount;default=1.0**)**
stop(interval**)**
[**Options: orient** ("horizon -tal", "vertical"), **length**, **mode**(determinate or indeterminate), **maximum** (default=100), **value** (current value), **variable** (name linked to value), **phase**(determinate: amt complete; indeterminate: complete a cycle when value increases by max) (start, step stop)]
Radiobutton
no new widgets
Scale
no new widgets
Scrollbar
no new widgets
Separator
Option: orient
no new widgets
Sizegrip
no new widgets
"southeast" resize only
Spinbox
get() ↳ current value
set() sets current value
<Up> & <Down> keys generate <<increment>> and <<decrement>> virtual event
[Options: from, to, increment, values, wrap, format, command]
Text
no new widgets
Toplevel
no new widgets
Treeview
[Options: columns (list of ids), **displaycolumns** (list or "#all"), **height** (rows visible), **padding**, **selectmode**("extended", "browse", "none"), **show** ("tree", "headings")**]**
bbox(*item*, *column=None*) ↳ (x, y, width, height)
get_children(*item=None*) list belonging to item if spec -ified otherwise root children
set_children (*item*, **newc hildren*) Replaces *item*'s child with *newchildren*
column (*column*, *option=N one*, ***kw*) Query or modify options; the valid options / values are:
id: ↳ the col name
anchor: standard value
minwidth: in pixels
stretch: True/False if the widget is resized
width: inpixels
To configure the tree column, call this with column = "#0"
delete(**items*) *items* and all their descendants
detach(**items*) Unlinks *items* from the tree - items may be reinserted at another point in the tree, but will not be displayed
exists (*item*) ↳ True if the *item* is present in the tree
focus(*item=None*) If *item* is specified set focus, otherwise ↳ the current focus item, or '' if there is none
heading (*column*, *option=No ne*, ***kw*) query/modify heading opts for the specified *col.* If *kw* is not given; ↳ a dict of the heading option values If *option* is specified then the value for that *option* is returned, Otherwise sets the options to the correspond ing values. The valid options/ values are:
text: text to display in the column heading
image: imageName to display to the right of the column heading
anchor: specifies how the heading text should be aligned - One of the standard Tk anchor values
command: callback to be invoked when the heading label is pressed
To configure the tree column heading, call this with column = "#0"
identify(*component*, *x*, *y*) ↳ a description of *component* under x,y or the empty string if no such *component* is pre- sent at that position
identify_row(*y*) ↳ the item ID of the item at position *y*
identify_column(*x*) ↳ the data column identifier of the cell at position *x*
The tree column has ID #0
identify_region(*x*, *y*) ↳ one of: **heading, separator, tree area** or **data cell**
identify_element(*x*, *y*) ↳ the element at position *x*, *y*
index(*item*) ↳ the integer index of *item* within its parent's list of children
insert (*parent*, *index*, *iid=No ne*, ***kw*) Creates a new item, complex, see explain- ation at https:// docs.python .org/3/library/tkinter.ttk.html #tkinter.ttk.Treeview.xview
item (*item*, *option=None*, *** kw*) Query or modify the options for the specified *item*
move(*item*, *parent*, *index*) Moves *item* to position *index* in *parent*'s list of children
next(*item*) ↳ the identifier of *item*'s next sibling, or '' if *item* is the last child of its parent
parent(*item*) ↳ the ID of the parent of *item*, or '' if *item* is at the top level of the hierarchy
prev(*item*) ↳ the identifier of *item*'s previous sibling, or '' if *item* is the first child of its parent
reattach (*item*, *parent*, *inde x*) = Treeviewmove()
see(*item*) Ensure *item* visible
selection() ↳ a tuple of selected items. For changing the selection state use the following selection methods
selection_set(**items*) *items* becomes new selection
selection_add(**items*) Add *items* to the selection
selection_remove(**items*) Remove *items* from selection
selection_toggle(**items*) Toggle the selection state of each item in *items*
set (*item*, *column=None*, *val ue=None*) One argument: ↳ a dictionary of column/value pairs for item. Two argu- ments: ↳ the current value of the specified column. Three arguments, sets value of given *column* in given *item* to the specified *value.*
tag_bind (*tagname*, *sequen ce=None*, *callback=None*) Bind a callback for the given event *sequence* to the tag *tagname* When an event is delivered to an item, the callbacks for each of the item's tags option are called
tag_configure (*tagname*, *option=None*, ***kw*) Query or modify the options for the specified *tagname;* If *kw* is not given, ↳ a dict of the option settings for *tagname;* If *option* is specified, ↳ the value for that *option* for the specified *tagname;* Other- wise, sets the options to the corresponding values for the given *tagname*
tag_has (*tagname*, *item=No ne*) If *item* is specified, ↳ 1 or 0 depending on whether the specified *item* has the given *tagname* Otherwise, ↳ a list of all items that have the specified tag
xview(*args) Query or modify horizontal position of the treeview
yview(**args*) Query or modify vertical position of the treeview

Compliments of: BIG DADDY &
www.wikipython.com

tkinter 2021 Toolbox Reference **Section 3:** Miscellaneous

white	snow	linen	grey1	red
orange	yellow	green	blue	navy
medium purple	coral1	Dark Orange3	turquoise4	DarkGoldenrod4
DarkGoldenrod2	gold2	yellow	LightGoldenrod2	green yellow
dark green	sea green	PaleGreen2	saddle brown	Dodger Blue3
blue2	SkyBlue1	SteelBlue4	VioletRed4	magenta
orchid3	purple	Dark Orchid4	thistle1	lavender blush
misty rose	grey10	grey25	grey50	grey75

BIG DADDY'S

© 2021 John A. Oakey V

JR2021D 16

PYTHON TOOLBOX For 3.5

tkinter 0221 Toolbox Reference

Common Color Definitions

Primary Colors:
Blue #0000FF (0,0,255)
Red #FF0000 (255,0,0)
Yellow #FFFF00 (255,255,0)
Orange #FF6600 (255,102,0)
Green #00FF00 (0,255,0)
Purple #6600FF (102,0,255)

Black #000000 (0,0,0)
White #FFFFFF (255...)

GA Tech: gold: (238, 179, 16) black: (35, 31, 32)
Georgia: red: (238, 45, 36) black: (0,0,0)
Vanderbilt gold: (186, 140, 12) black: (0,0,0)

GA State blue: (0, 57, 166) red:(204, 0, 0) black: (0,0,0)
Mercer orange(247,104,0)
USA USA **Blue** #3C3B6E (60,59,110) USA **Red** #B22234 (178,34,52) USA
Gold #d3af37 (83,69,22)
White #FFFFFF (3x255)

Army green: (29,32,12)
Navy blue: (3,38,44) gold: (232,176,15)
Air Force blue: (0,48,135) silver (138,141,143)
Marine scarlet (192,0,0) gold(140,123,33)
Cost Guard blue:(0,107,166)

What about tix? *Deprecated since 3.6:* unmaintained and should not be used in new code. Use tkinter.ttk instead.

Other GUI Modules

tkinter is not the only game in town, just the best supported. Over 40 "other" GUI "frame-works" with a few popular names like wxPython, PyQt (multiple vers), PySide, Kivy, Pygame are offered. An excellent summary of GUI modules can be found at **https://wikipython.org/moin/GuiProgramming**; *Note: not related to* ***www.wikipython.com*** *which creates and offers this toolbox.* Of the few we have tried, a newer entry seems poised to eclipse the rest based on ease of use. For new or intermediate users who want **high production** in a **reasonable time frame** (no GUI can offer a short time frame by its inherent nature) we suggest:

PySimpleGUI ("PSG") - has versions that "wrap" **tkinter** and three other GUI platforms. PySimpleGUI's **GitHub Stats** are **rated A++!** PSG enables a programmer to build custom GUI layouts in **minutes** in a **few lines of code**. It is easy for beginners, powerful enough for advanced users. It has extensive documentation and examples with many built-in color themes. PSG is based on the idea that a graphic layout can be simply defined as a **list** of rows named **"layout"** with each value of layout being a **list** of widgets, called **elements,** and their attributes. **A list of lists.**

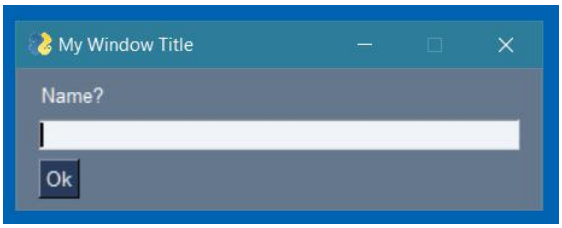
To download and install the module: **"pip3 install PySimpleGUI";** to use it just **"import PySimpleGUI as sg"** - that's it. Again, note that **widgets** in PSG are called **"elements"** and *virtually any imaginable widget is supported.*
PSG has just **5 steps to make a simple app**: *(and note… no classes and no callbacks, just simple pythonic code)*

(1) LAYOUT: Begin by defining a series of **lists** with each **row** of your window being **one list:**
i.e. syntax: **layout** = **[** [element1,(attributes)], [element2,(attributes)]…[] **]**
ex: **layout=[[sg.Text("Name?")],[sg.Input()],[sg.Button('Ok')]]** ← 3 rows with 3 one element lists
(2) CREATE A WINDOW *ex:* window = sg.**Window**("My Window Title", **layout**) ← creates your window
(3) IMPLEMENT AN EVENT LOOP *ex:* event, values = window. read() ← this event triggers and ↳ input
(4) PROCESS AND REACT— evaluate and/or process data and react
(5) CLOSE *ex:* window.close()

So in just **six lines of code** (on Windows) **this**:

```
import PySimpleGUI as sg
layout =[[sg.Text("Name?")], [sg.Input()],[sg.Button('Ok')]]
window = sg.Window("My Window Title", layout)
event, values = window.read()
print('Hello', values[0], "! Thanks.")
window.close()
```

produces this, gets the **input**, and prints a **response**



PySimpleGUI is not a great gimmick and a limited set of tkinter features—all of tkinter is there underneath and available if you want it, but widget creation, callback code, and recursive efforts to create a layout are done for you. The following widgets and features are available right out of the box according to the PSG docs which you can find at:
https://pysimplegui.readthedocs.io/en/latest/#a-complete-pysimplegui-program-getting-the-gist

- Text
- Single Line Input
- Button types:
  - File(s) Browse
  - Folder Browse
  - SaveAs
  - Normal button that returns an event
  - Close window
  - Realtime
  - Calendar chooser
  - Color chooser
  - Button Menu
- "Normal" TK Buttons
- Checkboxes
- Radio Buttons
- Listbox
- Option Menu
- Menubar
- Button Menu
- Slider
- Spinner
- Dial
- Graph
- Frame with title
- Icons
- Multi-line Text Input
- Scroll-able Output
- Images
- Tables
- Trees
- Progress Bar Async/Non-Blocking Windows
- Tabbed windows
- Paned windows
- Persistent Windows
- Multiple Windows - Unlimited number of windows can be open at the same time
- Redirect Python Output/Errors to scrolling window
- 'Higher level' APIs (like MessageBox, YesNobox)
- Single-Line-Of-Code Progress Bar & Debug Print
- Complete control of colors, look and feel
- Selection of pre-defined palettes
- Button images
- Horizontal and Vertical Separators
- Return values as dictionary
- Set focus
- Bind return key to buttons
- Group widgets into a column and place into window anywhere
- Scrollable columns
- Keyboard low-level key capture
- Mouse scroll-wheel support
- Get Listbox values as they are selected
- Get slider, spinner, combo as they are changed
- Update elements in a live window
- Bulk window-fill operation
- Save/Load from disk
- Borderless (no titlebar) windows (very classy looking)
- Always on top windows
- Menus with ALT-hotkey
- Right click pop-up menu
- Tooltips
- Clickable text
- Transparent windows
- Movable windows
- Animated GIFs
- No async programming required (no callbacks to worry about)
- Built-in debugger and REPL
- **User expandable by accessing underlying GUI Framework widgets directly.**

Compliments of: BIG DADDY & www.wikipython.com

Full Disclosure: This is a review—**not an ad**. The author of this toolbox and the staff of www.wikipython.com are unpaid hobbists and have no interest, financial or otherwise, in PySimpleGUI. In short, PSG is just a **better mouse-trap** for **most** users, in our opinion.