Python Tkinter

Introduction

Tk was developed as a GUI extension for the Tcl scripting language by John Ousterhout. The first release was in 1991. Tk proved as extremely successful in the 1990's, because it is easier to learn and to use than other toolkits. So it is no wonder that many programmers wanted to use Tk independently of Tcl. That's why bindings for lots of other programming languages have been developed, including Perl, Ada (called TASH), Python (called Tkinter), Ruby, and Common Lisp.

Tk provides the following widgets:

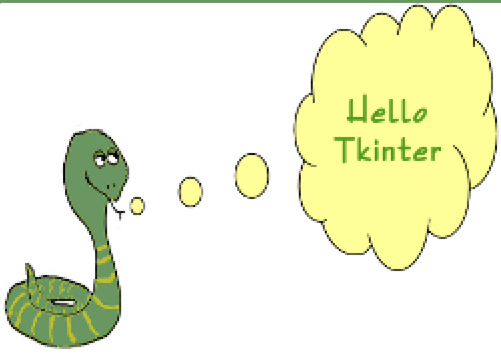
- button
- canvas
- checkbutton
- combobox
- entry
- frame
- label
- labelframe
- listbox
- menu
- menubutton
- message
- notebook
- tk_optionMenu
- panedwindow
- progressbar
- radiobutton
- scale
- scrollbar
- separator
- sizegrip
- spinbox
- text
- treeview

It provides the following top-level windows:

- tk_chooseColor - pops up a dialog box for the user to select a color.
- tk_chooseDirectory - pops up a dialog box for the user to select a directory.
- tk_dialog - creates a modal dialog and waits for a response.
- tk_getOpenFile - pops up a dialog box for the user to select a file to open.
- tk_getSaveFile - pops up a dialog box for the user to select a file to save.
- tk_messageBox - pops up a message window and waits for a user response.
- tk_popup - posts a popup menu.
- toplevel - creates and manipulates toplevel widgets.

Tkinter

Hello Tkinter Label



We will start our tutorial with one of the easiest widgets of Tk (Tkinter), i.e. a label. A Label is a Tkinter Widget class, which is used to display text or an image. The label is a widget that the user just views but not interact with.

There is hardly any book or introduction into a programming language, which doesn't start with the "Hello World" example. We will draw on tradition but will slightly modify the output to "Hello Tkinter" instead of "Hello World".

The following Python script uses Tkinter to create a window with the text "Hello Tkinter". You can use the Python interpretor to type this script line after line, or you can save it in a file, for example "hello.py":

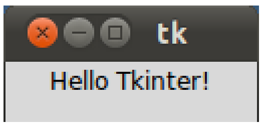
```
import tkinter as tk


# if you are still working under a Python 2 version,
# comment out the previous line and uncomment the following line
# import Tkinter as tk


root = tk.Tk()


w = tk.Label(root, text="Hello Tkinter!")
w.pack()


root.mainloop()
```

Starting our example

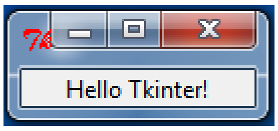
If we save the script under the name hello.py, we can start it like this using the command shell:

```
$ python3 hello.py
```

If you run the command under the Gnome and Linux, the window the window will look like this:



Under Windows it appears in the Windows look and feel:



Explanation

The tkinter module, containing the Tk toolkit, has always to be imported. In our example, we imported tkinter by renaming it into tk, which is the preferred way to do it:

```
import tkinter as tk
```

To initialize tkinter, we have to create a Tk root widget, which is a window with a title bar and other decoration provided by the window manager. The root widget has to be created before any other widgets and there can only be one root widget.

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

The next line of code contains the Label widget. The first parameter of the Label call is the name of the parent window, in our case "root". So our Label widget is a child of the root widget. The keyword parameter "text" specifies the text to be shown:

```
w = tk.Label(root, text="Hello Tkinter!")
```

The pack method tells Tk to fit the size of the window to the given text.

```
w.pack()
```

The window won't appear until we enter the Tkinter event loop:

```
root.mainloop()
```

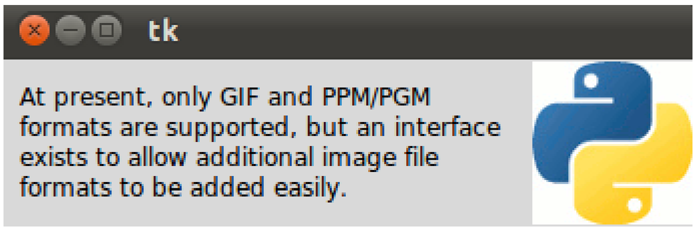
Our script will remain in the event loop until we close the window.

Using Images in Labels

As we have already mentioned, labels can contain text and images. The following example contains two labels, one with a text and the other one with an image.

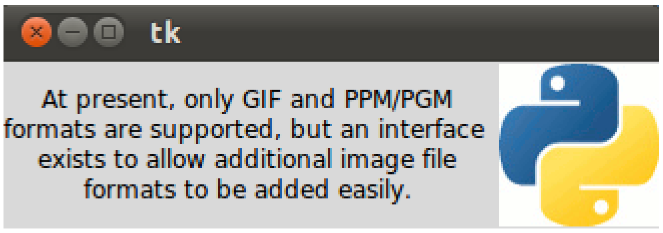
```
import tkinter as tk


root = tk.Tk()

logo = tk.PhotoImage(file="python_logo_small.gif")


w1 = tk.Label(root, image=logo).pack(side="right")


explanation = """At present, only GIF and PPM/PGM

formats are supported, but an interface

exists to allow additional image file

formats to be added easily."""


w2 = tk.Label(root,
              justify=tk.LEFT,
              padx = 10,
              text=explanation).pack(side="left")
root.mainloop()
```

If you start this script, it will look like this using Ubuntu Linux with Gnome desktop:



At present, only GIF and PPM/PGM
formats are supported, but an interface
exists to allow additional image file
formats to be added easily.

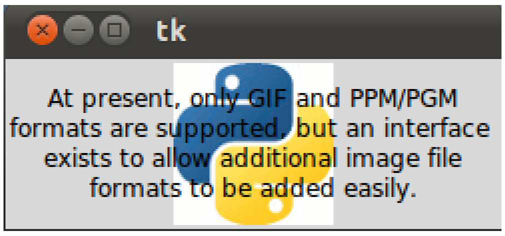
The "justify" parameter can be used to justify a text on the LEFT, RIGHT or CENTER. padx can be used to add additional horizontal padding around a text label. The default padding is 1 pixel. pady is similar for vertical padding. The previous example without justify (default is centre) and padx looks like this:



At present, only GIF and PPM/PGM
formats are supported, but an interface
exists to allow additional image file
formats to be added easily.

You want the text drawn on top of the image? No problem! We need just one label and use the image and the text option at the same time. By default, if an image is given, it is drawn instead of the text. To get the text as well, you have to use the compound option. If you set the compound option to CENTER the text will be drawn on top of the image:

```
import tkinter as tk

root = tk.Tk()
logo = tk.PhotoImage(file="python_logo_small.gif")

explanation = """At present, only GIF and PPM/PGM
formats are supported, but an interface
exists to allow additional image file
formats to be added easily."""

w = tk.Label(root,
             compound = tk.CENTER,
             text=explanation,
             image=logo).pack(side="right")

root.mainloop()
```



At present, only GIF and PPM/PGM formats are supported, but an interface exists to allow additional image file formats to be added easily.

We can have the image on the right side and the text left justified with a padding of 10 pixel on the left and right side by changing the Label command like this:

```
w = Label(root,
          justify=LEFT,
          compound = LEFT,
          padx = 10,
          text=explanation,
          image=logo).pack(side="right")
```

If the compound option is set to BOTTOM, LEFT, RIGHT, or TOP, the image is drawn correspondingly to the bottom, left, right or top of the text.

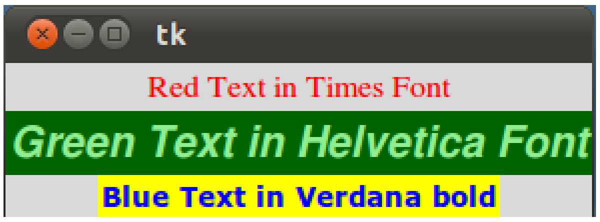
Colorized Labels in various fonts

Some Tk widgets, like the label, text, and canvas widget, allow you to specify the fonts used to display text. This can be achieved by setting the attribute "font". typically via a "font" configuration option. You have to consider that fonts are one of several areas that are not platform-independent.

The attribute fg can be used to have the text in another colour and the attribute bg can be used to change the background colour of the label.

```
import tkinter as tk

root = tk.Tk()
tk.Label(root,
               text="Red Text in Times Font",
               fg = "red",
               font = "Times").pack()
tk.Label(root,
               text="Green Text in Helvetica Font",
               fg = "light green",
               bg = "dark green",
               font = "Helvetica 16 bold italic").pack()
tk.Label(root,
               text="Blue Text in Verdana bold",
               fg = "blue",
                bg = "yellow",
                font = "Verdana 10 bold").pack()


root.mainloop()
```

The result looks like this:



Red Text in Times Font

Green Text in Helvetica Font

Blue Text in Verdana bold

Message Widget

The widget can be used to display short text messages. The message widget is similar in its functionality to the Label widget, but it is more flexible in displaying text, e.g. the font can be changed while the Label widget can only display text in a single font. It provides a multiline object, that is the text may span more than one line. The text is automatically broken into lines and justified. We were ambiguous, when we said, that the font of the message widget can be changed. This means that we can choose arbitrarily a font for one widget, but the text of this widget will be rendered solely in this font. This means that we can't change the font within a widget. So it's not possible to have a text in more than one font. If you need to display text in multiple fonts, we suggest to use a Text widget.

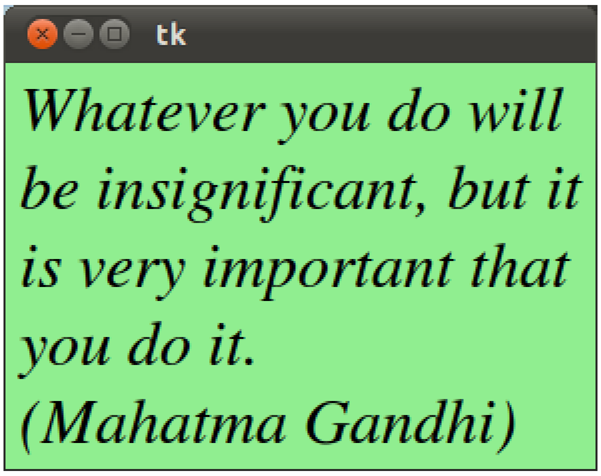
The syntax of a message widget:

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

Let's have a look at a simple example. The following script creates a message with a famous saying by Mahatma Gandhi:

```
import tkinter as tk
master = tk.Tk()
whatever_you_do = "Whatever you do will be insignificant, but it is very important
that you do it.\n(Mahatma Gandhi)"
msg = tk.Message(master, text = whatever_you_do)
msg.config(bg='lightgreen', font=('times', 24, 'italic'))
msg.pack()
tk.mainloop()
```

The widget created by the script above looks like this:



Whatever you do will be insignificant, but it is very important that you do it.
(Mahatma Gandhi)

The Options in Detail

Option	Meaning
anchor	The position, where the text should be placed in the message widget: N, NE, E, SE, S, SW, W, NW, or CENTER. The Default is CENTER.
aspect	Aspect ratio, given as the width/height relation in percent. The default is 150, which means that the message will be 50% wider than it is high. Note that if the width is explicitly set, this option is ignored.
background	The background color of the message widget. The default value is system specific.
bg	Short for background.
borderwidth	Border width. Default value is 2.
bd	Short for borderwidth.
cursor	Defines the kind of cursor to show when the mouse is moved over the message widget. By default the standard cursor is used.
font	Message font. The default value is system specific.
foreground	Text color. The default value is system specific.
fg	Same as foreground.
highlightbackground	Together with highlightcolor and highlightthickness, this option controls how to draw the highlight region.
highlightcolor	See highlightbackground.
highlightthickness	See highlightbackground.

justify	Defines how to align multiple lines of text. Use LEFT, RIGHT, or CENTER. Note that to position the text inside the widget, use the anchor option. Default is LEFT.
padx	Horizontal padding. Default is -1 (no padding).
pady	Vertical padding. Default is -1 (no padding).
relief	Border decoration. The default is FLAT. Other possible values are SUNKEN, RAISED, GROOVE, and RIDGE.
takefocus	If true, the widget accepts input focus. The default is false.
text	Message text. The widget inserts line breaks if necessary to get the requested aspect ratio. (text/Text)
textvariable	Associates a Tkinter variable with the message, which is usually a StringVar. If the variable is changed, the message text is updated.
width	Widget width given in character units. A suitable width based on the aspect setting is automatically chosen, if this option is not given.

Tkinter Buttons

The Button widget is a standard Tkinter widget, which is used for various kinds of buttons. A button is a widget which is designed for the user to interact with, i.e. if the button is pressed by mouse click some action might be started. They can also contain text and images like labels. While labels can display text in various fonts, a button can only display text in a single font. The text of a button can span more than one line.

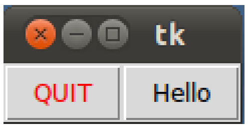
A Python function or method can be associated with a button. This function or method will be executed, if the button is pressed in some way.

Example for the Button Class

The following script defines two buttons: one to quit the application and another one for the action, i.e. printing the text "Tkinter is easy to use!" on the terminal.

```
import tkinter as tk

def write_slogan():

    print("Tkinter is easy to use!")


root = tk.Tk()

frame = tk.Frame(root)

frame.pack()


button = tk.Button(frame,
                   text="QUIT",
                   fg="red",
                   command=quit)
button.pack(side=tk.LEFT)
slogan = tk.Button(frame,
                   text="Hello",
                   command=write_slogan)
slogan.pack(side=tk.LEFT)

root.mainloop()
```

The result of the previous example looks like this:



Variable Classes

Some widgets (like text entry widgets, radio buttons and so on) can be connected directly to application variables by using special options: variable, textvariable, onvalue, offvalue, and value. This connection works both ways: if the variable changes for any reason, the widget it's connected to will be updated to reflect the new value. These Tkinter control variables are used like regular Python variables to keep certain values. It's not possible to hand over a regular Python variable to a widget through a variable or textvariable option. The only kinds of variables for which this works are variables that are subclassed from a class called Variable, defined in the Tkinter module. They are declared like this:

- x = StringVar() # Holds a string; default value ""
- x = IntVar() # Holds an integer; default value 0
- x = DoubleVar() # Holds a float; default value 0.0
- x = BooleanVar() # Holds a boolean, returns 0 for False and 1 for True

To read the current value of such a variable, call the method get(). The value of such a variable can be changed with the set() method.

Radio Buttons

A radio button, sometimes called option button, is a graphical user interface element of Tkinter, which allows the user to choose (exactly) one of a predefined set of options. Radio buttons can contain text or images. The button can only display text in a single font. A Python function or method can be associated with a radio button. This function or method will be called, if you press this radio button.

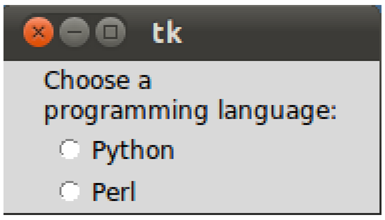
Radio buttons are named after the physical buttons used on old radios to select wave bands or preset radio stations. If such a button was pressed, other buttons would pop out, leaving the pressed button the only pushed in button.

Each group of Radio button widgets has to be associated with the same variable. Pushing a button changes the value of this variable to a predefined certain value.

Simple Example With Radio Buttons

```
import tkinter as tk

root = tk.Tk()

v = tk.IntVar()

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

root.mainloop()
print(v.get())
```

The result of the previous example looks like this:



Choose a
programming language:
Python
Perl