Sliders

Introduction

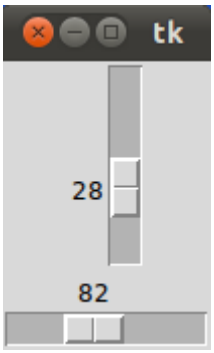
A slider is a Tkinter object with which a user can set a value by moving an indicator. Sliders can be vertically or horizontally arranged. A slider is created with the Scale method().

Using the Scale widget creates a graphical object, which allows the user to select a numerical value by moving a knob along a scale of a range of values. The minimum and maximum values can be set as parameters, as well as the resolution. We can also determine if we want the slider vertically or horizontally positioned. A Scale widget is a good alternative to an Entry widget, if the user is supposed to put in a number from a finite range, i.e. a bounded numerical value.

A Simple Example

```
from tkinter import *


master = Tk()
w = Scale(master, from_=0, to=42)
w.pack()
w = Scale(master, from_=0, to=200, orient=HORIZONTAL)
w.pack()


mainloop()
```

If we start this script, we get a window with a vertical and a horizontal slider:



Accessing Slider Values

We have demonstrated in the previous example how to create sliders. But it's not enough to have a slider, we also need a method to query it's value. We can accomplish this with the get method. We extend the previous example with a Button to view the values. If this button is pushed, the values of both sliders is printed into the terminal from which we have started the script:

```

```

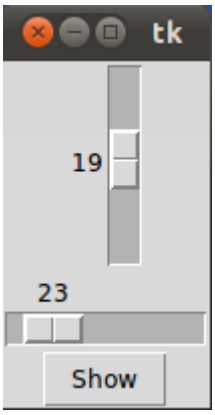
```
from tkinter import *


def show_values():
    print (w1.get(), w2.get())


master = Tk()
w1 = Scale(master, from_=0, to=42)
w1.pack()
w2 = Scale(master, from_=0, to=200, orient=HORIZONTAL)
w2.pack()
Button(master, text='Show', command=show_values).pack()


mainloop()
```

Initializing Sliders

A slider starts with the minimum value, which is 0 in our examples. There is a way to initialize Sliders with the set(value) method:

```
from tkinter import *


def show_values():
    print (w1.get(), w2.get())


master = Tk()
w1 = Scale(master, from_=0, to=42)
w1.set(19)
w1.pack()
w2 = Scale(master, from_=0, to=200, orient=HORIZONTAL)
w2.set(23)
w2.pack()
Button(master, text='Show', command=show_values).pack()


mainloop()
```

The previous script creates the following window, if it is called:

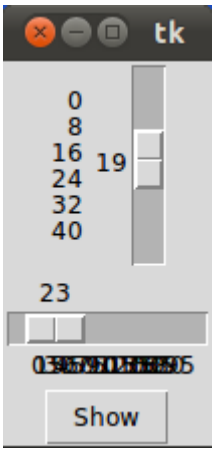


tickinterval and length

If the option tickinterval is set to a number, the ticks of the scale will be displayed as multiples of that value. We add a tickinterval to our previous example.

```
from tkinter import *


def show_values():
    print (w1.get(), w2.get())


master = Tk()
w1 = Scale(master, from_=0, to=42, tickinterval=8)
w1.set(19)
w1.pack()
w2 = Scale(master, from_=0, to=200,tickinterval=10, orient=HORIZONTAL)
w2.set(23)
w2.pack()
Button(master, text='Show', command=show_values).pack()


mainloop()
```

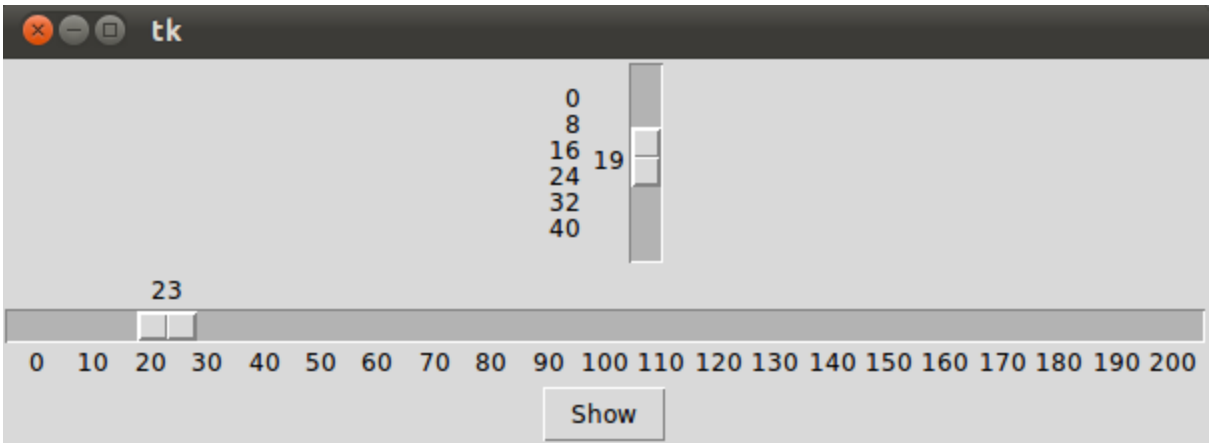
If we start this program, we recognize that the vertical slider has the values 0, 8, 16, 24, 32, 40 added to its left side. The horizontal slider has also the numbers 0,10,20, 30, ..., but we can't see them, because the get smeared on top of each other, because the slider is not long enough:



To solve this problem we have to increase the length of our horizontal slider. We set the option length. length defines the x dimension, if the scale is horizontal and the y dimension, if the scale is vertical. So we change the definition of w2 in the following way:

```
w2 = Scale(master, from_=0, to=200, length=600,tickinterval=10,
orient=HORIZONTAL)
```

The result looks like this:



Text Widgets

Introduction and Simple Examples

A text widget is used for multi-line text area. The Tkinter text widget is very powerful and flexible and can be used for a wide range of tasks. Though one of the main purposes is to provide simple multi-line areas, as they are often used in forms, text widgets can also be used as simple text editors or even web browsers.

Furthermore, text widgets can be used to display links, images, and HTML, even using CSS styles.

In most other tutorials and text books, it's hard to find a very simple and basic example of a text widget. That's why we want to start our chapter with a such an example:

We create a text widget by using the Text() method. We set the height to 2, i.e. two lines and the width to 30, i.e. 30 characters. We can apply the method insert() on the object T, which the Text() method had returned, to include text. We add two lines of text.

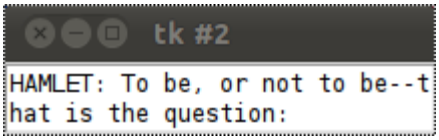
```
from tkinter import *


root = Tk()
T = Text(root, height=2, width=30)
T.pack()
T.insert(END, "Just a text Widget\nin two lines\n")
mainloop()
```

The result should not be very surprising:

Let's change our little example a tiny little bit. We add another text, the beginning of the famous monologue from Hamlet:

```
from tkinter import *


root = Tk()
T = Text(root, height=2, width=30)
T.pack()
quote = """HAMLET: To be, or not to be--that is the question:
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune
Or to take arms against a sea of troubles
And by opposing end them. To die, to sleep--
No more--and by a sleep to say we end
The heartache, and the thousand natural shocks
That flesh is heir to. 'Tis a consummation
Devoutly to be wished."""
T.insert(END, quote)
mainloop()
```

If we start our little script, we get a very unsatisfying result. We can see in the window only the first line of the monologue and this line is even broken into two lines. We can see only two lines in our window, because we set the height to the value 2. Furthermpre, the width is set to 30, so Tkinter has to break the first line of the monologue after 30 characters.



One solution to our problem consists in setting the height to the number of lines of our monologue and set width wide enough to display the widest line completely.

But there is a better technique, which you are well acquainted with from your browser and other applications: scrolling

Here is the example of getting the contents of text entry.

```
from tkinter import *


def show_values():
    print (T.get("1.0",END))
```

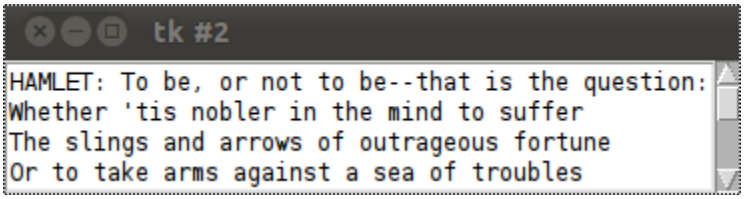
```
root = Tk()
S = Scrollbar(root)
T = Text(root, height=4, width=50)
S.pack(side=RIGHT, fill=Y)
T.pack(side=LEFT, fill=Y)
S.config(command=T.yview)
T.config(yscrollcommand=S.set)

Button(text='Show', command=show_values).pack()
mainloop()
```

Scrollbars

So let's add a scrollbar to our window. To this purpose, Tkinter provides the Scrollbar() method. We call it with the root object as the only parameter.

```
from tkinter import *
root = Tk()
S = Scrollbar(root)
T = Text(root, height=4, width=50)
S.pack(side=RIGHT, fill=Y)
T.pack(side=LEFT, fill=Y)
S.config(command=T.yview)
T.config(yscrollcommand=S.set)
quote = """HAMLET: To be, or not to be--that is the question:
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune
Or to take arms against a sea of troubles
And by opposing end them. To die, to sleep--
No more--and by a sleep to say we end
The heartache, and the thousand natural shocks
That flesh is heir to. 'Tis a consummation
Devoutly to be wished."""
T.insert(END, quote)
mainloop()
```

The result is a lot better. We have now always 4 lines in view, but all lines can be viewed by using the scrollbar on the right side of the window:

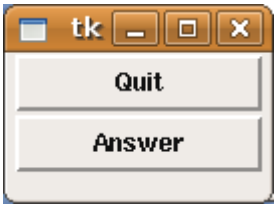


Dialogues and Message Boxes

Introduction

Tkinter (and TK of course) provides a set of dialogues (dialogs in American English spelling), which can be used to display message boxes, showing warning or errors, or widgets to select files and colours. There are also simple dialogues, asking the user to enter string, integers or float numbers.

Let's look at a typical GUI Session with Dialogues and Message boxes. There might be a button starting the dialogue, like the "quit" button in the following window:



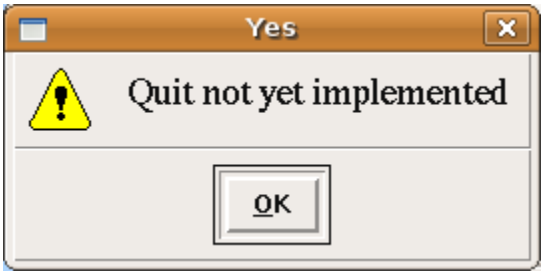
Python script, which implements the previous example:

```
from tkinter import *
from tkinter import messagebox


def answer():
    messagebox.showerror("Answer", "Sorry, no answer available")


def callback():
    if messagebox.askyesno('Verify', 'Really quit?'):
        messagebox.showwarning('Yes', 'Not yet implemented')
    else:
```

```
        messagebox.showinfo('No', 'Quit has been cancelled')


Button(text='Quit', command=callback).pack(fill=X)
Button(text='Answer', command=answer).pack(fill=X)
mainloop()
```

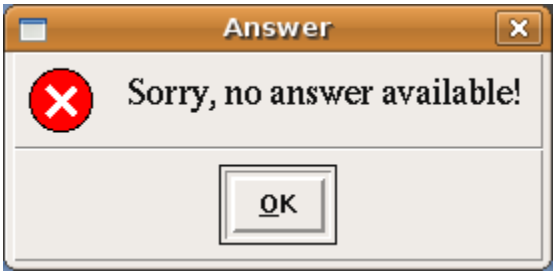
Pushing the "quit" button raises the Verify window:

Let's assume that we want to warn users that the "quit" functionality is not yet implemented. In this case we can use the warning message to inform the user, if he or she pushes the "yes" button:



If somebody types the "No" button, the "Cancel" message box is raised:

Let's go back to our first Dialogue with the "quit" and "answer" buttons. If the "Answer" functionality is not implemented, it might be useful to use the following error message box:



Message Boxes

The message dialogues are provided by the messagebox module.

The messagebox consists of the following functions, which correspond to dialog windows:

- askokcancel(title=None, message=None, **options)
  Ask if operation should proceed; return true if the answer is ok
- askquestion(title=None, message=None, **options)
  Ask a question
- askretrycancel(title=None, message=None, **options)
  Ask if operation should be retried; return true if the answer is yes
- askyesno(title=None, message=None, **options)
  Ask a question; return true if the answer is yes
- askyesnocancel(title=None, message=None, **options)
  Ask a question; return true if the answer is yes, None if cancelled.
- showerror(title=None, message=None, **options)
  Show an error message
- showinfo(title=None, message=None, **options)
  Show an info message
- showwarning(title=None, message=None, **options)
  Show a warning message

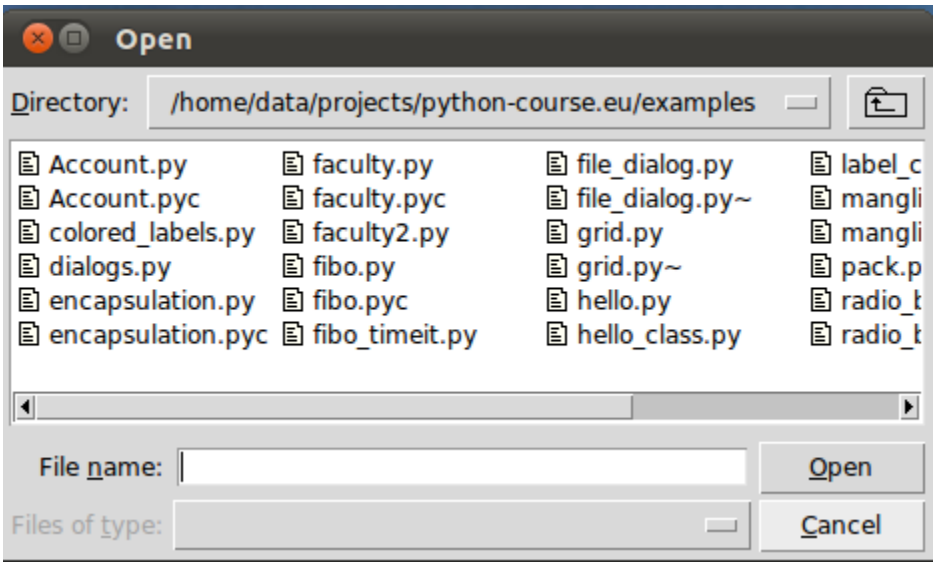
Open File Dialogue

There is hardly any serious application, which doesn't need a way to read from a file or write to a file. Furthermore, such an application might have to choose a directory. Tkinter provides the module tkFileDialog for these purposes.

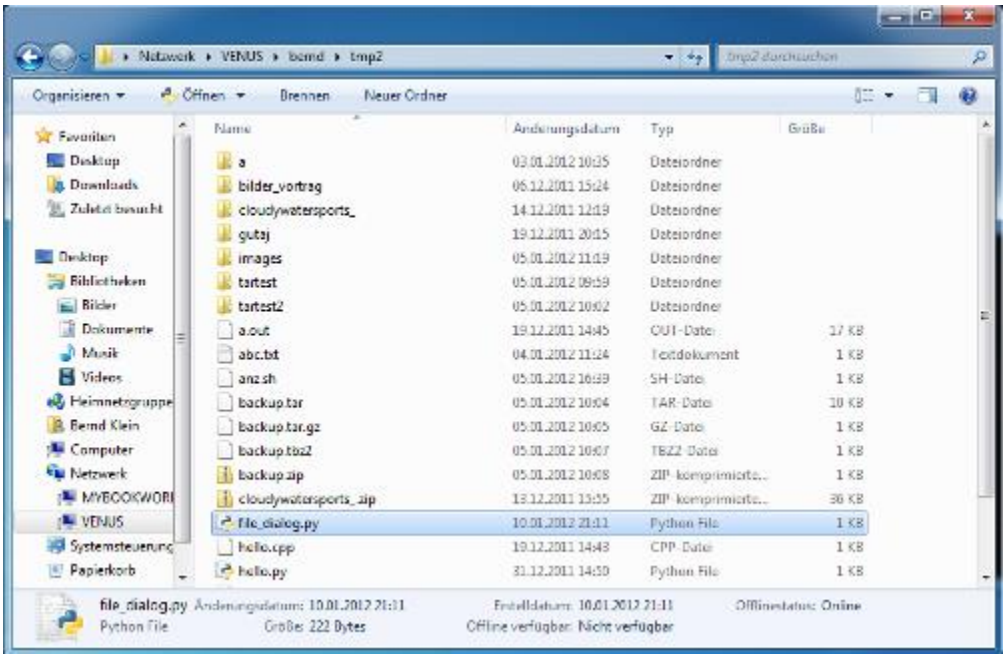
```
from tkinter import *
from tkinter import filedialog


def callback():
    name= filedialog.askopenfilename()
```

```
    print(name)


master = Tk()
Button(text='File Open', command=callback).pack(fill=X)
mainloop()
```

The code above creates a window with a single button with the text "File Open". If the button is pushed, the following window appears:



The look-and-feel of the file-open-dialog depends on the GUI of the operating system. The above example was created using a gnome desktop under Linux. If we start the same program under Windows 7, it looks like this:



Choosing a Colour

There are applications where the user should have the possibility to select a colour. Tkinter provides a pop-up menu to choose a colour. To this purpose we have to import the tkColorChooser module and have to use the method askColor:

```
result = tkColorChooser.askColor ( color, option=value, ...)
```

If the user clicks the OK button on the pop-up window, respectively, the return value of askColor() is a tuple with two elements, both a representation of the chosen colour, e.g. ((106, 150, 98), '#6a9662')
The first element return[0] is a tuple (R, G, B) with the RGB representation in decimal values (from 0 to 255). The second element return[1] is a hexadecimal representation of the chosen colour.
If the user clicks "Cancel" the method returns the tuple (None, None).

The optional keyword parameters are:

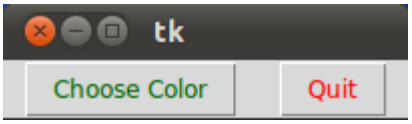
color	The variable color is used to set the default colour to be displayed. If color is not set, the initial colour will be grey.

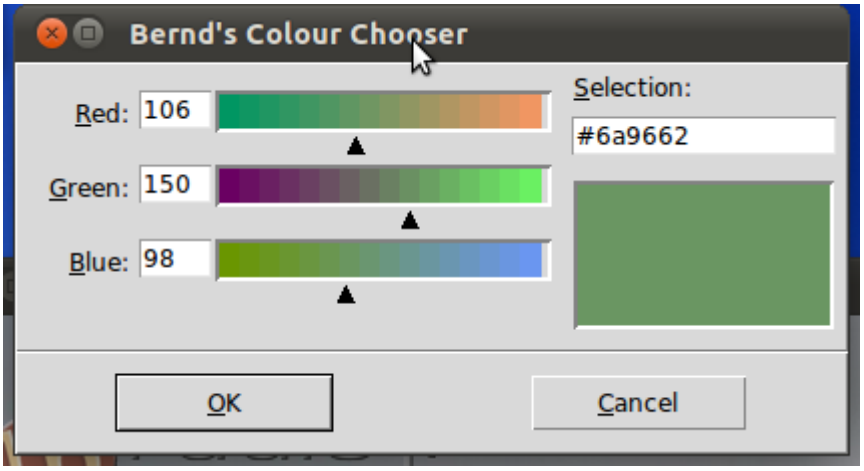
title	The text assigned to the variable title will appear in the pop-up window's title area. The default title is "Color".
parent	Make the pop-up window appear over window W. The default behaviour is that it appears over the root window.

Let's have a look at an example:

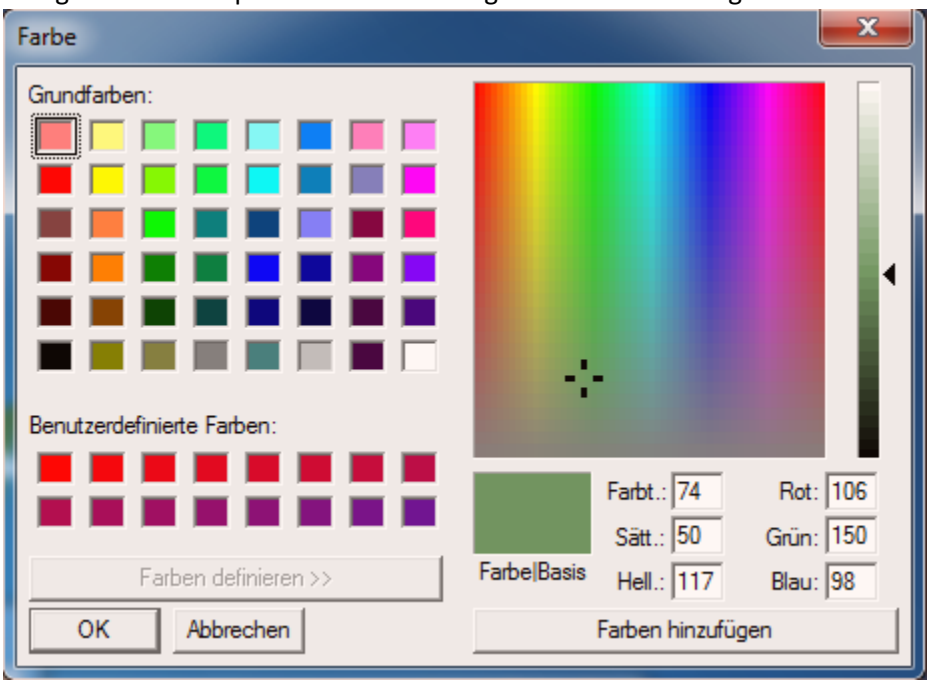
```
from tkinter import *
from tkinter import colorchooser


def callback():
    result = colorchooser.askcolor(color="#6A9662", title = "Choose Color")
    print("RGB:" + str(result[0]))
    print("hex:" + str(result[1]))


root = Tk()
Button(text='Choose Color',
fg="darkgreen",command=callback).pack(side=LEFT, padx=10)
Button(text='Quit',command=quit,fg="red").pack(side=LEFT, padx=10)
mainloop()
```

The look and feel depends on the operating system (e.g. Linux or Windows) and the chosen GUI (GNOME, KDE and so on). The following windows appear, if you use Gnome:





Using the same script under Windows 7 gives us the following result:



Menus

Introduction

Most people, if confronted with the word "menu", will immediately think of a menu in a restaurant. Even though the menu of a restaurant and the menu of a computer program have at first glance nothing in common, we can see that yet the have a lot in common. In a restaurant, a menu is a presentation of all their food and beverage offerings, while in a computer application it presents all the commands and functions of the application, which are available to the user via the graphical user interface.

Menus in GUIs are presented with a combination of text and symbols to represent the choices. Selecting with the mouse (or finger on touch screens) on one of the symbols or text, an action will be started. Such an action or operation can, for example, be the opening or saving of a file, or the quitting or exiting of an application.

A context menu is a menu in which the choices presented to the user are modified according to the current context in which the user is located.

We introduce in this chapter of our Python Tkinter tutorial the pull-down menus of Tkinter, i.e.

the lists at the top of the windows, which appear (or pull down), if you click on an item like, for example "File", "Edit" or "Help".

A Simple Menu Example

The following Python script creates a simple application window with menus.

```
from tkinter import *
from tkinter import messagebox

def donothing():
    messagebox.showwarning('Yes', 'Not yet implemented')

root = Tk()

## create menu bar
menubar = Menu(root)
##file menu
filemenu = Menu(menubar,tearoff=0)
menubar.add_cascade(label = "File", menu = filemenu)
filemenu.add_command(label="New", command = donothing)
filemenu.add_command(label = "Open", command = donothing)
filemenu.add_command(label = "Save", command = donothing)
filemenu.add_command(label = "Save as...", command = donothing)
filemenu.add_command(label = "Close", command = donothing)
filemenu.add_separator()
filemenu.add_command(label = "Exit", command = root.quit)

##edit menu
editmenu = Menu(menubar,tearoff=0)
menubar.add_cascade(label = "Edit", menu = editmenu)
editmenu.add_command(label = "Undo", command = donothing)
editmenu.add_separator()
editmenu.add_command(label = "Cut", command = donothing)
editmenu.add_command(label = "Copy", command = donothing)
editmenu.add_command(label = "Paste", command = donothing)
editmenu.add_command(label = "Delete", command = donothing)
editmenu.add_command(label = "Select All", command = donothing)

##help menu
helpmenu = Menu(menubar,tearoff=0)
menubar.add_cascade(label = "Help", menu = helpmenu)
helpmenu.add_command(label = "Help Index", command = donothing)
helpmenu.add_command(label = "About...", command = donothing)


root.config(menu = menubar)
mainloop()
```

It looks like this, if started: