Chapter 17

GUI Programming III

This chapter contains a few more GUI odds and ends.

17.1 Title bar

The GUI window that Tkinter creates says Tk by default. Here is how to change it:

```
root.title('Your title')
```

17.2 Disabling things

Sometimes you want to disable a button so it can't be clicked. Buttons have an attribute `state` that allows you to disable the widget. Use `state=DISABLED` to disable the button and `state=NORMAL` to enable it. Here is an example that creates a button that starts out disabled and then enables it:

```
button = Button(text='Hi', state=DISABLED, command=function)
button.configure(state=NORMAL)
```

You can use the `state` attribute to disable many other types of widgets, too.

17.3 Getting the state of a widget

Sometimes, you need to know things about a widget, like exactly what text is in it or what its background color is. The `cget` method is used for this. For example, the following gets the text of a label called `label`:

```
label.cget('text')
```

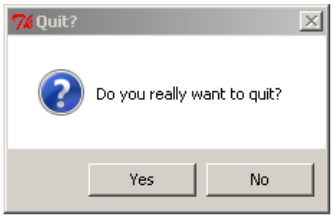
This can be used with buttons, canvases, etc., and it can be used with any of their properties, like `bg`, `fg`, `state`, etc. As a shortcut, Tkinter overrides the `[]` operators, so that `label['text']` accomplishes the same thing as the example above.

17.4 Message boxes

Message boxes are windows that pop up to ask you a question or say something and then go away. To use them, we need an import statement:

```
from tkinter.messagebox import *
```

There are a variety of different types of message boxes. For each of them you can specify the message the user will see as well as the title of the message box. Here are three types of message boxes, followed by the code that generates them:




```
showinfo(title='Message for you', message='Hi There!')
askquestion(title='Quit?', message='Do you really want to quit?')
showwarning(title='Warning', message='Unsupported format')
```

Below is a list of all the types of message boxes. Each displays a message in its own way.

Message Box	Special properties
`showinfo`	`OK` button
`askokcancel`	`OK` and `Cancel` buttons
`askquestion`	`Yes` and `No` buttons
`askretrycancel`	`Retry` and a `Cancel` buttons
`askyesnocancel`	`Yes`, `No`, and `Cancel` buttons
`showerror`	An error icon and an `OK` button
`showwarning`	A warning icon an an `OK` button

Each of these functions returns a value indicating what the user clicked. See the next section for a simple example of using the return value. Here is a table of the return values:

Function	Return value (based on what user clicks)
`showinfo`	Always returns `'ok'`
`askokcancel`	`OK`—**`True`** `Cancel` or window closed—**`False`**
`askquestion`	`Yes`—`'yes'` `No`—`'no'`
`askretrycancel`	`Retry`—**`True`** `Cancel` or window closed—**`False`**
`askyesnocancel`	`Yes`—**`True`** `No`—**`False`** anything else—**`None`**
`showerror`	Always returns `'ok'`
`showwarning`	Always returns `'ok'`

17.5 Destroying things

To get rid of a widget, use its `destroy` method. For instance, to get rid of a button called `button`, do the following:

```
button.destroy()
```

To get rid of the entire GUI window, use the following:

```
root.destroy()
```

Stopping a window from being closed When your user tries to close the main window, you may want to do something, like ask them if they really want to quit. Here is a way to do that:

```
from tkinter import *
from tkinter.messagebox import askquestion

def quitter_function():
    answer = askquestion(title='Quit?', message='Really quit?')
    if answer=='yes':
        root.destroy()

root = Tk()
root.protocol('WM_DELETE_WINDOW', quitter_function)
mainloop()
```

The key is the following line, which cause `quitter_function` to be called whenever the user tries to close the window.

```
root.protocol('WM_DELETE_WINDOW', quitter_function)
```

17.6 Updating

Tkinter updates the screen every so often, but sometimes that is not often enough. For instance, in a function triggered by a button press, Tkinter will not update the screen until the function is done.

If, in that function, you want to change something on the screen, pause for a short while, and then change something else, you will need to tell Tkinter to update the screen before the pause. To do that, just use this:

```
root.update()
```

If you only want to update a certain widget, and nothing else, you can use the `update` method of that widget. For example,

```
canvas.update()
```

A related thing that is occasionally useful is to have something happen after a scheduled time interval. For instance, you might have a timer in your program. For this, you can use the `after` method. Its first argument is the time in milliseconds to wait before updating and the second argument is the function to call when the time is right. Here is an example that implements a timer:

```
from time import time
from tkinter import *

def update_timer():
    time_left = int(90 - (time()-start))
    minutes = time_left // 60
    seconds = time_left % 60
    time_label.configure(text='{}:{:02d}'.format(minutes, seconds))
    root.after(100, update_timer)

root = Tk()
time_label = Label()
time_label.grid(row=0, column=0)

start = time()
update_timer()

mainloop()
```

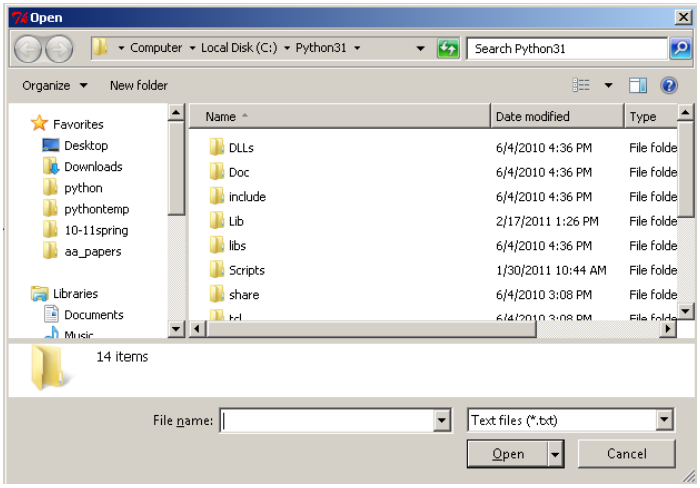
This example uses the `time` module, which is covered in Section 20.2.

17.7 Dialogs

Many programs have dialog boxes that allow the user to pick a file to open or to save a file. To use them in Tkinter, we need the following import statement:

```
from tkinter.filedialog import *
```

Tkinter dialogs usually look like the ones that are native to the operating system.



Here are the most useful dialogs:

Dialog	Description
`askopenfilename`	Opens a typical file chooser dialog
`askopenfilenames`	Like previous, but user can pick more than one file
`asksaveasfilename`	Opens a typical file save dialog
`askdirectory`	Opens a directory chooser dialog

The return value of `askopenfilename` and `asksaveasfilename` is the name of the file selected. There is no return value if the user does not pick a value. The return value of `askopenfilenames` is a list of files, which is empty if no files are selected. The `askdirectory` function returns the name of the directory chosen.

There are some options you can pass to these functions. You can set `initialdir` to the directory you want the dialog to start in. You can also specify the file types. Here is an example:

```
filename=askopenfilename(initialdir='c:\\python31\\',
                         filetypes=[('Image files', '.jpg .png .gif'),
                                    ('All files', '*')])
```

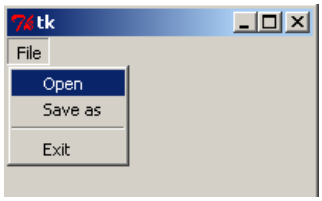
A short example Here is an example that opens a file dialog that allows you to select a text file. The program then displays the contents of the file in a textbox.

```
from tkinter import *
from tkinter.filedialog import *
from tkinter.scrolledtext import ScrolledText

root = Tk()
textbox = ScrolledText()
textbox.grid()

filename=askopenfilename(initialdir='c:\\python31\\',
                         filetypes=[('Text files', '.txt'),
```

```
                                        ('All files', '*')])
s = open(filename).read()
textbox.insert(1.0, s)

mainloop()
```

17.8 Menu bars

We can create a menu bar, like the one below, across the top of a window.



Here is an example that uses some of the dialogs from the previous section:

```
from tkinter import *
from tkinter.filedialog import *

def open_callback():
    filename = askopenfilename()
    # add code here to do something with filename

def saveas_callback():
    filename = asksaveasfilename()
    # add code here to do something with filename

root = Tk()
menu = Menu()
root.config(menu=menu)
file_menu = Menu(menu, tearoff=0)
file_menu.add_command(label='Open', command=open_callback)
file_menu.add_command(label='Save as', command=saveas_callback)
file_menu.add_separator()
file_menu.add_command(label='Exit', command=root.destroy)
menu.add_cascade(label='File', menu=file_menu)

mainloop()
```

17.9 New windows

Creating a new window is easy. Use the `Toplevel` function:

```
window = Toplevel()
```

You can add widgets to the new window. The first argument when you create the widget needs to be the name of the window, like below

```
new_window = Toplevel()
label = Label(new_window, text='Hi')
label.grid(row=0, column=0)
```

17.10 pack

There is an alternative to `grid` called `pack`. It is not as versatile as `grid`, but there are some places where it is useful. It uses an argument called `side`, which allows you to specify four locations for your widgets: `TOP`, `BOTTOM`, `LEFT`, and `RIGHT`. There are two useful optional arguments, `fill` and `expand`. Here is an example.

```
button1=Button(text='Hi')
button1.pack(side=TOP, fill=X)
button2=Button(text='Hi')
button2.pack(side=BOTTOM)
```

The `fill` option causes the widget to fill up the available space given to it. It can be either `X`, `Y` or `BOTH`. The `expand` option is used to allow the widget to expand when its window is resized. To enable it, use `expand=YES`.

Note You can use `pack` for some frames, and `grid` for others; just don't mix `pack` and `grid` within the same frame, or Tkinter won't know quite what to do.

17.11 StringVar

In Section 16.5 we saw how to tie a Tkinter variable, called an `IntVar`, to a check button or a radio button. Tkinter has another type of variable called a `StringVar` that holds strings. This type of variable can be used to change the text in a label or a button or in some other widgets. We already know how to change text using the `configure` method, and a `StringVar` provides another way to do it.

To tie a widget to a `StringVar`, use the `textvariable` option of the widget. A `StringVar` has `get` and `set` methods, just like an `IntVar`, and whenever you set the variable, any widgets that are tied to it are automatically updated.

Here is a simple example that ties two labels to the same `StringVar`. There is also a button that when clicked will alternate the value of the StringVar (and hence the text in the labels).

```
from tkinter import *

def callback():
    global count
    s.set('Goodbye' if count%2==0 else 'Hello')
    count +=1

root = Tk()

count = 0
s = StringVar()
s.set('Hello')

label1 = Label(textvariable = s, width=10)
label2 = Label(textvariable = s, width=10)
button = Button(text = 'Click me', command = callback)

label1.grid(row=0, column=0)
label2.grid(row=0, column=1)
button.grid(row=1, column=0)

mainloop()
```

17.12 More with GUIs

We have left out quite a lot about Tkinter. See Lundh's *Introduction to Tkinter* [2] for more. Tkinter is versatile and simple to work with, but if you need something more powerful, there are other third-party GUIs for Python.