Chapter 1

Creating the GUI Form and Adding Widgets

Content

- Creating our first Python GUI
- Preventing the GUI from being resized
- Adding a label to the GUI form
- Creating buttons and changing their text attributes
- Creating textbox widgets
- Setting the focus to a widget and disabling widgets
- Creating combo box widgets
- Creating a check button with different initial states
- Using radio button widgets
- Using scrolled text widgets
- Adding several widgets in a loop

Here is an overview of the Python modules (ending in a `.py` extension) for this chapter:

Creating our first Python GUI

Getting ready

To follow this recipe, a working Python development environment is a prerequisite. The IDLE GUI, which ships with Python, is enough to start. IDLE was built using `tkinter`!

How to do it…

Let's take a look at how to create our first Python GUI:

1. Create a new Python module and name it `First_GUI.py`.
2. At the top of the `First_GUI.py` module, import `tkinter`:

```
import tkinter as tk
```

3.Create an instance of the `Tk` class:

```
win = tk.Tk()
```

4.Use the instance variable to set a title:

```
win.title("Python GUI")
```

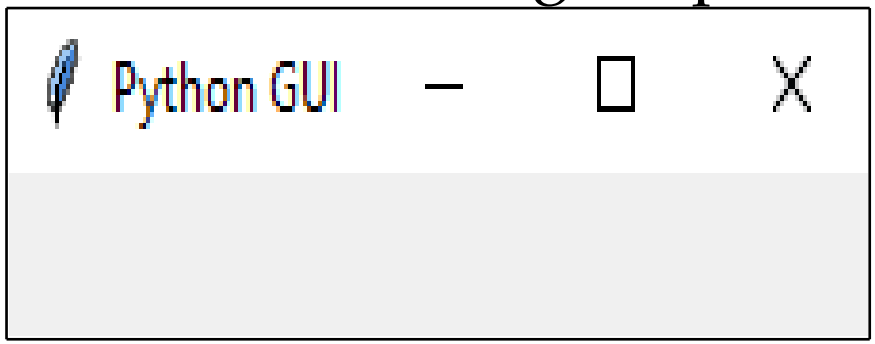
5.Start the window's main event loop:

```
win.mainloop()
```

The following screenshot shows the four lines of `First_GUI.py` required to create the resulting GUI:

```
#======================
# imports
#======================
import tkinter as tk

# Create instance
win = tk.Tk()

# Add a title
win.title("Python GUI")

#======================
# Start GUI
#======================
win.mainloop()
```

6. Run the GUI module. On executing the preceding code, the following output is obtained:



Preventing the GUI from being resized

By default, a GUI created using `tkinter` can be resized. This is not always ideal. The widgets we place onto our GUI forms might end up being resized in an improper way, so in this recipe, we will learn how to prevent our GUI from being resized by the user of our GUI application.

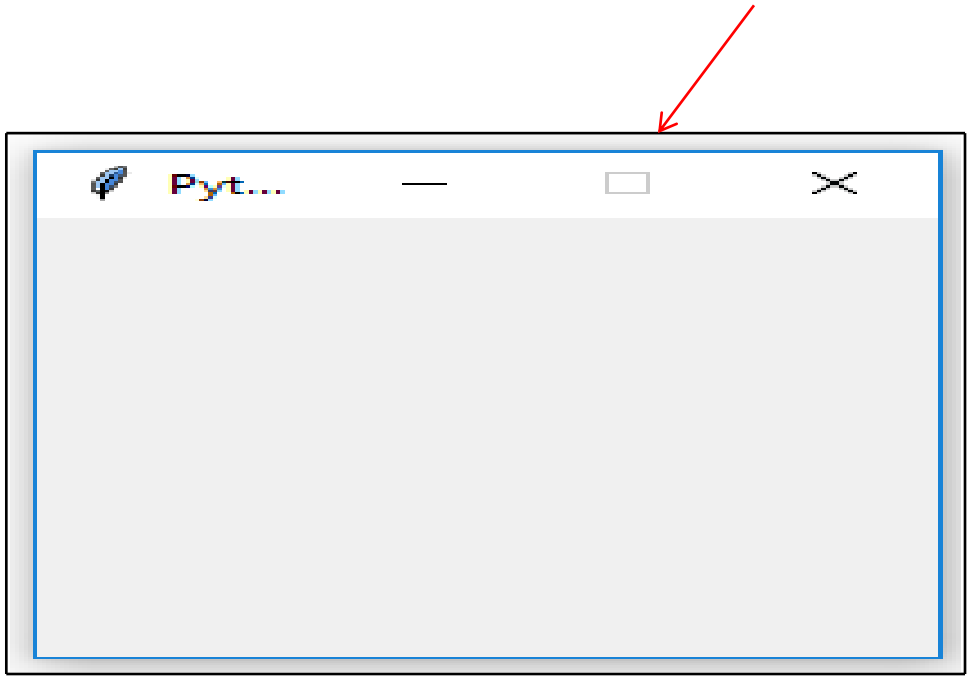
1. Start with the module from the previous recipe and save it as `Gui_not_resizable.py`.
2. Use the `Tk` instance variable, `win`, to call the `resizable` method:

```
win.resizable(False, False)
```

Here is the code to prevent the GUI from being resized (`GUI_not_resizable.py`):

```
#======================
# imports
#======================
import tkinter as tk

# Create instance
win = tk.Tk()

# Add a title
win.title("Python GUI")

# Disable resizing the GUI by passing in False/False
win.resizable(False, False)

# Enable resizing x-dimension, disable y-dimension
# win.resizable(True, False)

#======================
# Start GUI
#======================
win.mainloop()
```

3. Run the code. Running the code creates this GUI:



Adding a label to the GUI form

Getting ready

We are extending the first recipe, *Creating our first Python GUI*. We will leave the GUI resizable, so don't use the code from the second recipe (or comment the `win.resizable` line out).

Adding a label to the GUI form

How to do it…

Perform the following steps to add a label to the GUI from:

1. Start with the `First_GUI.py` module and save it as `GUI_add_label.py`.
2. Import `ttk`:

```
from tkinter import ttk
```

3. Use `ttk` to add a label:

```
ttk.Label(win, text="A Label")
```

Adding a label to the GUI form

How to do it…

Perform the following steps to add a label to the GUI from:

4. Use the grid layout manager to position the label:

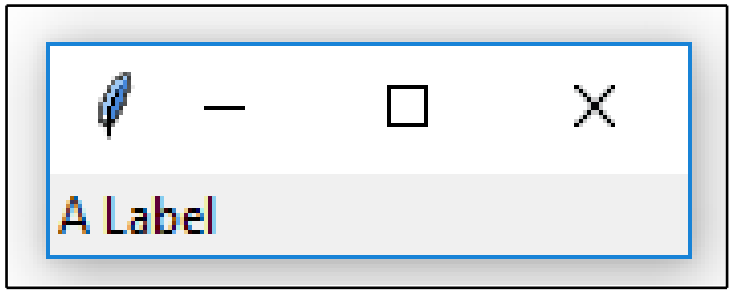
```
.grid(column=0, row=0)
```

In order to add a `Label` widget to our GUI, we will import the `ttk` module from `tkinter`.
Please note the two `import` statements on lines 9 and 10.

The following code is added just above `win.mainloop()`, which is located at the bottom of the first and second recipes (`GUI_add_label.py`):

```
#======================
# imports
#======================
import tkinter as tk
from tkinter import ttk

# Create instance
win = tk.Tk()

# Add a title
win.title("Python GUI")

# Adding a Label
ttk.Label(win, text="A Label").grid(column=0, row=0)

#======================
# Start GUI
#======================
win.mainloop()
```

5. Run the code and observe how a label is added to our GUI:



Creating buttons and changing their text attributes

In this recipe, we will add a button widget, and we will use this button to change an attribute of another widget that is a part of our GUI. This introduces us to callback functions and event handling in a Python GUI environment.

How to do it…

In this recipe, we will update the label we added in the previous recipe as well as the `text` attribute of the button. The steps to add a button that performs an action when clicked are as follows:

1. Start with the `GUI_add_label.py` module and save it as `GUI_create_button_change_property.py`.
2. Define a function and name it `click_me()`:

```
def click_me()
```

How to do it…

3. Use `ttk` to create a button and give it a `text` attribute:

```
action.configure(text="** I have been
Clicked! **")

a_label.configure (foreground='red')
a_label.configure(text='A Red Label')
```

How to do it…

4.Bind the function to the button:

```
action = ttk.Button(win, text="Click Me!", command=click_me)
```

5.Use the grid layout to position the button:

```
action.grid(column=1, row=0)
```

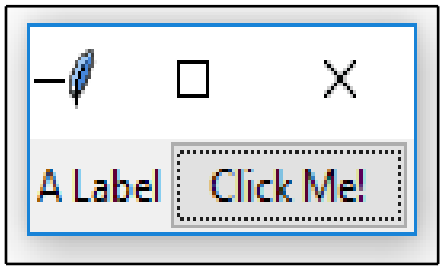
How to do it…

The preceding instructions produce the following code (`GUI_create_button_change_property.py`):

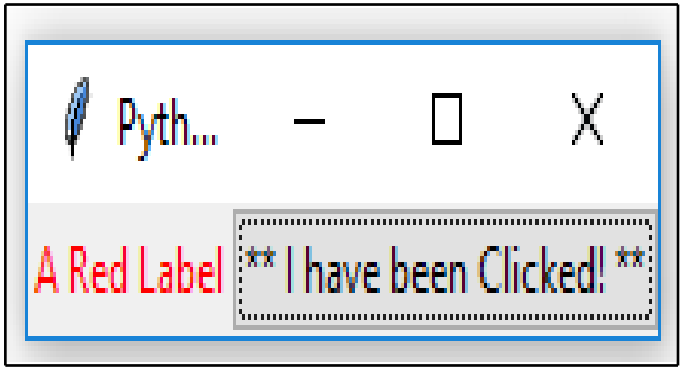
```
18 # Adding a Label that will get modified
19 a_label = ttk.Label(win, text="A Label")
20 a_label.grid(column=0, row=0)
21
22 # Button Click Event Function
23 def click_me():
24     action.configure(text="** I have been Clicked! **")
25     a_label.configure(foreground='red')
26     a_label.configure(text='A Red Label')
27
28 # Adding a Button
29 action = ttk.Button(win, text="Click Me!", command=click_me)
30 action.grid(column=1, row=0)
31
32 #======================
33 # Start GUI
34 #======================
35 win.mainloop()
```

6. Run the code and observe the output.

The following screenshot shows how our GUI looks before clicking the button:



After clicking the button, the color of the label changed and so did the text of the button, which can be seen in the following screenshot:



Creating textbox widgets

In `tkinter`, a typical one-line textbox widget is called `Entry`. In this recipe, we will add such an `Entry` widget to our GUI. We will make our label more useful by describing what the `Entry` widget is doing for the user.

Creating textbox widgets

How to do it…

Follow these steps to create textbox widgets:

1. Start with the `GUI_create_button_change_property.py` module and save it as `GUI_textbox_widget.py`.
2. Use the `tk` alias of `tkinter` to create a `StringVar` variable:

```
name = tk.StringVar()
```

Creating textbox widgets

How to do it…

Follow these steps to create textbox widgets:

3. Create a `ttk.Entry` widget and assign it to another variable:

```
name_entered = ttk.Entry(win,
width=12, textvariable=name)
```

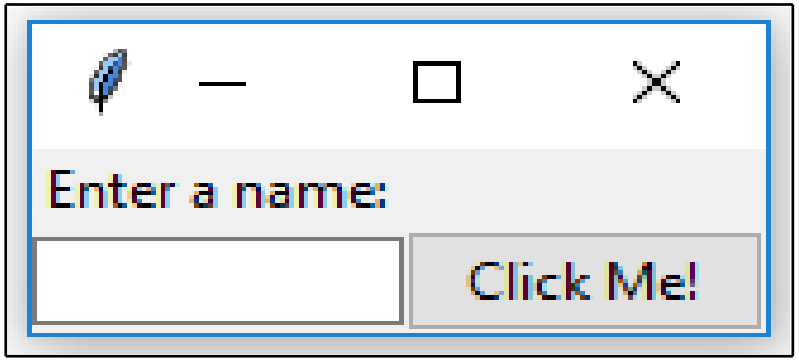
4. Use this variable to position the `Entry` widget:

```
name_entered.grid(column=0, row=1)
```

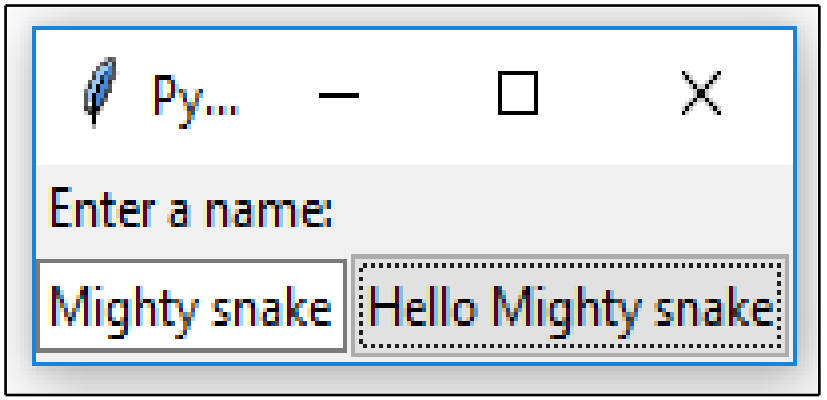
The preceding instructions produce the following code (`GUI_textbox_widget.py`):

```
22 # Modified Button Click Function
23 def click_me():
24     action.configure(text='Hello ' + name.get())
25
26 # Changing our Label
27 ttk.Label(win, text="Enter a name:").grid(column=0, row=0)
28
29 # Adding a Text box Entry widget
30 name = tk.StringVar()
31 name_entered = ttk.Entry(win, width=12, textvariable=name)
32 name_entered.grid(column=0, row=1)
```

5. Run the code and observe the output; our GUI looks like this:



6. Enter some text and click the button; we will see that there is a change in the GUI, which is as follows:



Setting the focus to a widget and disabling widgets

While our GUI is nicely improving, it would be more convenient and useful to have the cursor appear in the Entry widget as soon as the GUI appears.

In this recipe, we learn how to make the cursor appear in the Entry box for immediate text Entry rather than the need for the user to *click* into the Entry widget to give it the `focus` method before typing into the entry widget.

Setting the focus to a widget and disabling widgets

Getting ready

This recipe extends the previous recipe, *Creating textbox widgets*. Python is truly great. All we have to do to set the focus to a specific control when the GUI appears is call the `focus()` method on an instance of a `tkinter` widget we previously created. In our current GUI example, we assigned the `ttk.Entry` class instance to a variable named `name_entered`.

Now, we can give it the focus.

How to do it…

Place the following code just above the previous code, which is located at the bottom of the module, and which starts the main window's event loop, like we did in the previous recipes:

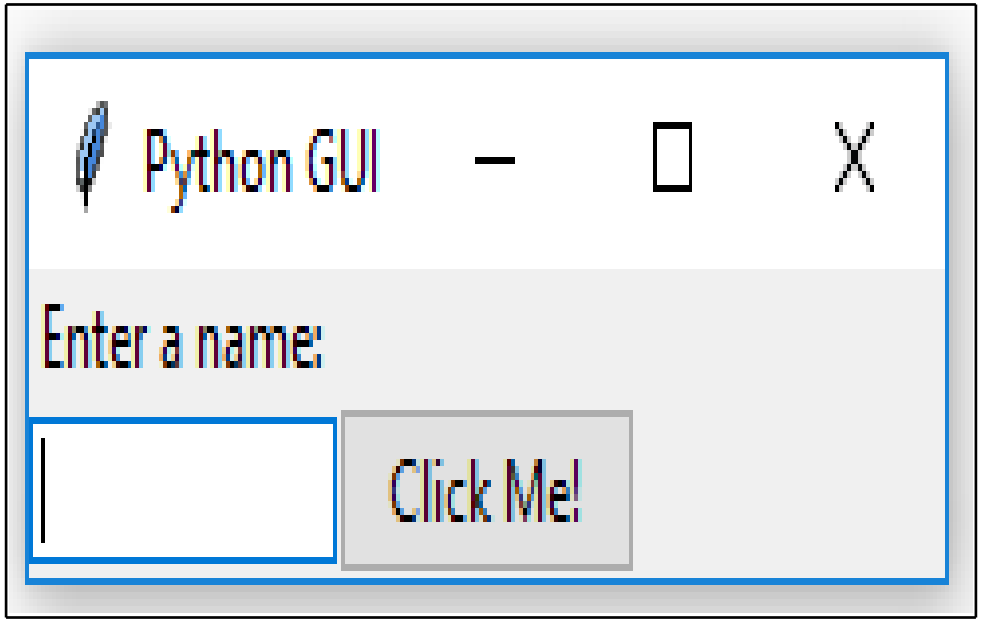
1. Start with the `GUI_textbox_widget.py` module and save it as `GUI_set_focus.py`.
2. Use the `name_entered` variable we assigned the `ttk Entry` widget instance to and call the `focus()` method on this variable:

```
name_entered.focus()
```

The preceding instructions produce the following code (`GUI_set_focus.py`):

```
29 # Adding a Textbox Entry widget
30 name = tk.StringVar()
31 name_entered = ttk.Entry(win, width=12, textvariable=name)
32 name_entered.grid(column=0, row=1)
33
34 # Adding a Button
35 action = ttk.Button(win, text="Click Me!", command=click_me)
36 action.grid(column=1, row=1)
37
38 name_entered.focus()      # Place cursor into name Entry
39 #======================
40 # Start GUI
41 #======================
42 win.mainloop()
```

3. Run the code and observe the output.



To disable widgets, we will set an attribute on the widget. We can make the button disabled by adding the following code below line 37 of the Python code to create the button:

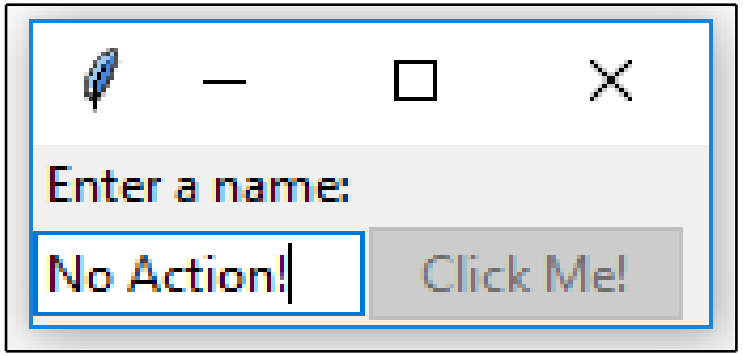
1. Use the `GUI_set_focus.py` module and save it as `GUI_disable_button_widget.py`.
2. Use the `action` button variable to call the `configure` method and set the `state` attribute to `disabled`:
   `action.configure(state='disabled')`
3. Call the `focus()` method on the `name_entered` variable:

```
name_entered.focus()
```

The preceding instructions produce the following code (`GUI_disable_button_widget.py`):

```
34 # Adding a Button
35 action = ttk.Button(win, text="Click Me!", command=click_me)
36 action.grid(column=1, row=1)
37 action.configure(state='disabled')    # Disable the Button Widget
38
39 name_entered.focus()      # Place cursor into name Entry
```

4. Run the code. After adding the preceding line of Python code, clicking the button no longer creates an action:



Creating combobox widgets

In this recipe, we will improve our GUI by adding drop-down comboboxes that can have initial default values. While we can restrict the user to only certain choices, we can also allow the user to type in whatever they wish.

Getting ready

This recipe extends the previous recipe, *Setting the focus to a widget and disabling widgets*.

Creating combobox widgets

How to do it…

We insert another column between the Entry widget and the `Button` widget using the grid layout manager. Here is the Python code:

1. Start with the `GUI_set_focus.py` module and save it as `GUI_combobox_widget.py`.
2. Change the button column to `2`:

```
action = ttk.Button(win, text="Click
Me!", command=click_me)
action.grid(column=2, row=1)
```

Creating combobox widgets

How to do it…

We insert another column between the Entry widget and the `Button` widget using the grid layout manager. Here is the Python code:

3. Create a new `ttk.Label` widget:

```
ttk.Label(win, text="Choose a
number:").grid(column=1, row=0)
```

4.Create a new `ttk.Combobox` widget:

```
number_chosen = ttk.Combobox(win, width=12,
textvariable=number)
```

5.Assign values to the `Combobox` widget:

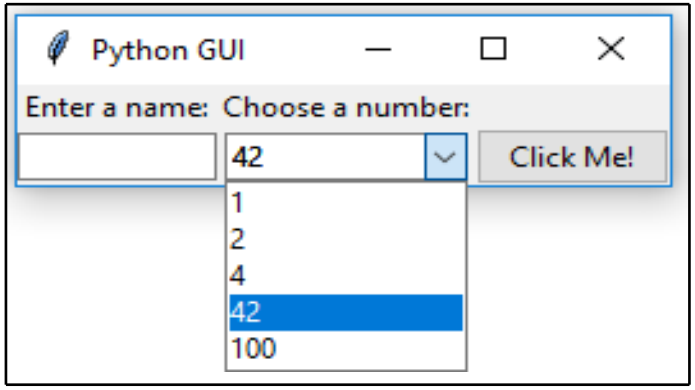
```
number_chosen['value'] = (1, 2, 4, 42, 100)
```

6.Place the `Combobox` widget into `column 1`:

```
number_chosen.grid(colum=1,row=1)
number_chosen.current(0)
```

```
31 # Adding a Textbox Entry widget
32 name = tk.StringVar()
33 name_entered = ttk.Entry(win, width=12, textvariable=name)
34 name_entered.grid(column=0, row=1)                          # column 0
35
36 # Adding a Button
37 action = ttk.Button(win, text="Click Me!", command=click_me)
38 action.grid(column=2, row=1)                                # <= change column to 2
39
40 ttk.Label(win, text="Choose a number:").grid(column=1, row=0)
41 number = tk.StringVar()
42 number_chosen = ttk.Combobox(win, width=12, textvariable=number)
43 number_chosen['values'] = (1, 2, 4, 42, 100)
44 number_chosen.grid(column=1, row=1)                         # <= Combobox in column 1
45 number_chosen.current(0)
46
47 name_entered.focus()      # Place cursor into name Entry
48 #======================
49 # Start GUI
50 #======================
51 win.mainloop()
```

7. Run the code.



If we want to restrict the user to only being able to select the values we have programmed into the `Combobox` widget, we can do it by passing the `state` attribute into the constructor. Modify *line 42* as follows:

1. Start with the `GUI_combobox_widget.py` module and save it as `GUI_combobox_widget_readonly.py`.
2. Set the `state` attribute when creating the `Combobox` widget:

```
number_chosen = ttk.Combobox(win,
width=12, textvariable=number,
state='readonly')
```

The preceding steps produce the following code (`GUI_combobox_widget_readonly.py`:

```
40 ttk.Label(win, text="Choose a number:").grid(column=1, row=0)
41 number = tk.StringVar()
42 number_chosen = ttk.Combobox(win, width=12, textvariable=number, state='readonly')
43 number_chosen['values'] = (1, 2, 4, 42, 100)
44 number_chosen.grid(column=1, row=1)
45 number_chosen.current(0)
```

3. Run the code.

Now, users can no longer type values into the `Combobox` widget.

We can display the value chosen by the user by adding the following line of code to our button click event callback function:

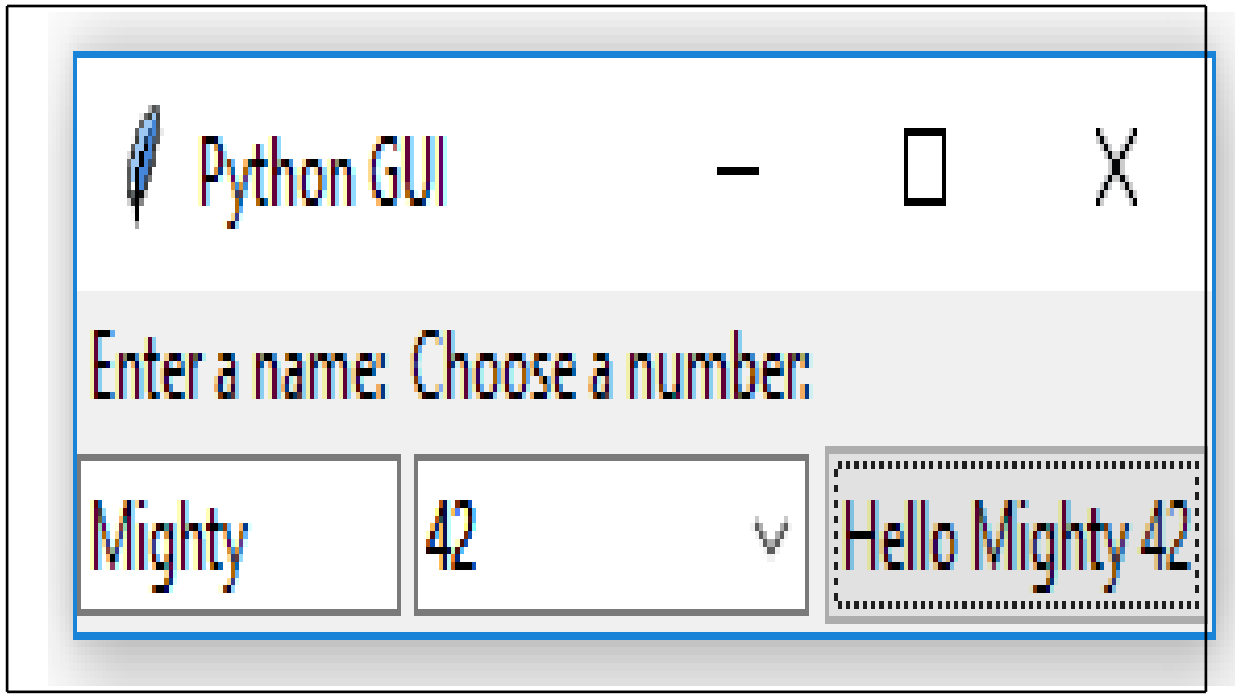
1. Start with the `GUI_combobox_widget_readonly.py` module and save it as `GUI_combobox_widget_readonly_plus_display_number.py`.
2. Extend the button click event handler by using the `get()` method on the `name` variable, use concatenation (`+ ' ' +`), and also get the number from the `number_chosen` variable (also calling the `get()` method on it):

```
def click_me():
  action.configure(text='Hello ' +
  name.get() + ' ' +  number_chosen.get())
```

3. Run the code.

After choosing a number, entering a name, and then clicking the button, we get the following GUI result, which now also displays the number selected next to the name entered

(`GUI_combobox_widget_readonly_plus_display_number.py`:



Creating a check button with different initial states

In this recipe, we will add three check button widgets, each with a different initial state:

- The first is disabled and has a checkmark in it. The user cannot remove this checkmark as the widget is disabled.
- The second check button is enabled, and by default has no checkmark in it, but the user can click it to add a checkmark.
- The third check button is both enabled and checked by default. The users can uncheck and recheck the widget as often as they like

Getting ready

This recipe extends the previous recipe, *Creating combobox widgets*.

How to do it…

Here is the code for creating three check button widgets that differ in their states:

1. Start with the `GUI_combobox_widget_readonly_plus_display_number.py` module and save it as `GUI_checkbutton_widget.py`.
2. Create three `tk.IntVar` instances and save them in local variables:

```
chVarDis = tk.IntVar()
chVarUn = tk.IntVar()
chVarEn = tk.IntVar()
```

How to do it…

Here is the code for creating three check button widgets that differ in their states:

3. Set the `text` attributes for each of the `Combobox` widgets we are creating:

```
text="Disabled"
text="UnChecked"
text="Enabled"
```

4. Set their `state` to `deselect`/`select`:

```
check1.select()
check2.deselect()
check3.select()
```

How to do it…

Here is the code for creating three check button widgets that differ in their states:

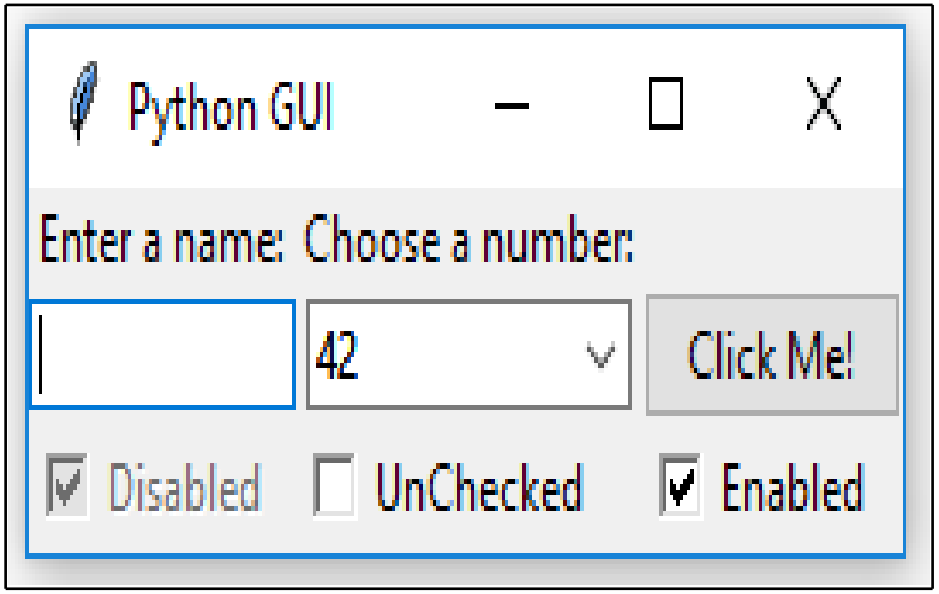
5. Use `grid` to lay them out:

```
check1.grid(column=0, row=4, sticky=tk.W)
check2.grid(column=1, row=4, sticky=tk.W)
check3.grid(column=2, row=4, sticky=tk.W)
```

The preceding steps will finally produce the following code (`GUI_checkbutton_widget.py`):

```
35  # Adding a Button
36  action = ttk.Button(win, text="Click Me!", command=click_me)
37  action.grid(column=2, row=1)
38
39  # Creating a label and a Combobox
40  ttk.Label(win, text="Choose a number:").grid(column=1, row=0)
41  number = tk.StringVar()
42  number_chosen = ttk.Combobox(win, width=12, textvariable=number, state='readonly')
43  number_chosen['values'] = (1, 2, 4, 42, 100)
44  number_chosen.grid(column=1, row=1)
45  number_chosen.current(0)
46  # Creating three checkbuttons
47  chVarDis = tk.IntVar()
48  check1 = tk.Checkbutton(win, text="Disabled", variable=chVarDis, state='disabled')
49  check1.select()
50  check1.grid(column=0, row=4, sticky=tk.W)
51
52  chVarUn = tk.IntVar()
53  check2 = tk.Checkbutton(win, text="UnChecked", variable=chVarUn)
54  check2.deselect()
55  check2.grid(column=1, row=4, sticky=tk.W)
56
57  chVarEn = tk.IntVar()
58  check3 = tk.Checkbutton(win, text="Enabled", variable=chVarEn)
59  check3.select()
60  check3.grid(column=2, row=4, sticky=tk.W)
61
62  name_entered.focus()      # Place cursor into name Entry
63  #======================
64  # Start GUI
65  #======================
66  win.mainloop()
```

6. Run the module. Running the new code results in the following GUI:



Using radio button widgets

In this recipe, we will create three radio button widgets. We will also add some code that changes the color of the main form, depending upon which radio button is selected.

Getting ready

This recipe extends the previous recipe, Creating a check button with different initial states.

How to do it…

We add the following code to the previous recipe:

1. Start with the `GUI_checkbutton_widget.py` module and save it as `GUI_radiobutton_widget.py`.
2. Create three module-level global variables for the color names:

```
COLOR1 = "Blue"
COLOR2 = "Gold"
COLOR3 = "Red"
```

How to do it…

We add the following code to the previous recipe:

3. Create a callback function for the radio buttons:

```
if radSel == 1:
        win.configure(background=COLOR1)
elif radSel == 2:
        win.configure(background=COLOR2)
elif radSel == 3:
        win.configure(background=COLOR3)
```

How to do it…

We add the following code to the previous recipe:

4. Create three `tk` radio buttons:

```
rad1 = tk.Radiobutton(win, text=COLOR1,
variable=radVar, value=1, command=radCall)
rad2 = tk.Radiobutton(win, text=COLOR2,
variable=radVar, value=2, command=radCall)
rad3 = tk.Radiobutton(win, text=COLOR3,
variable=radVar, value=3, command=radCall)
```

How to do it…

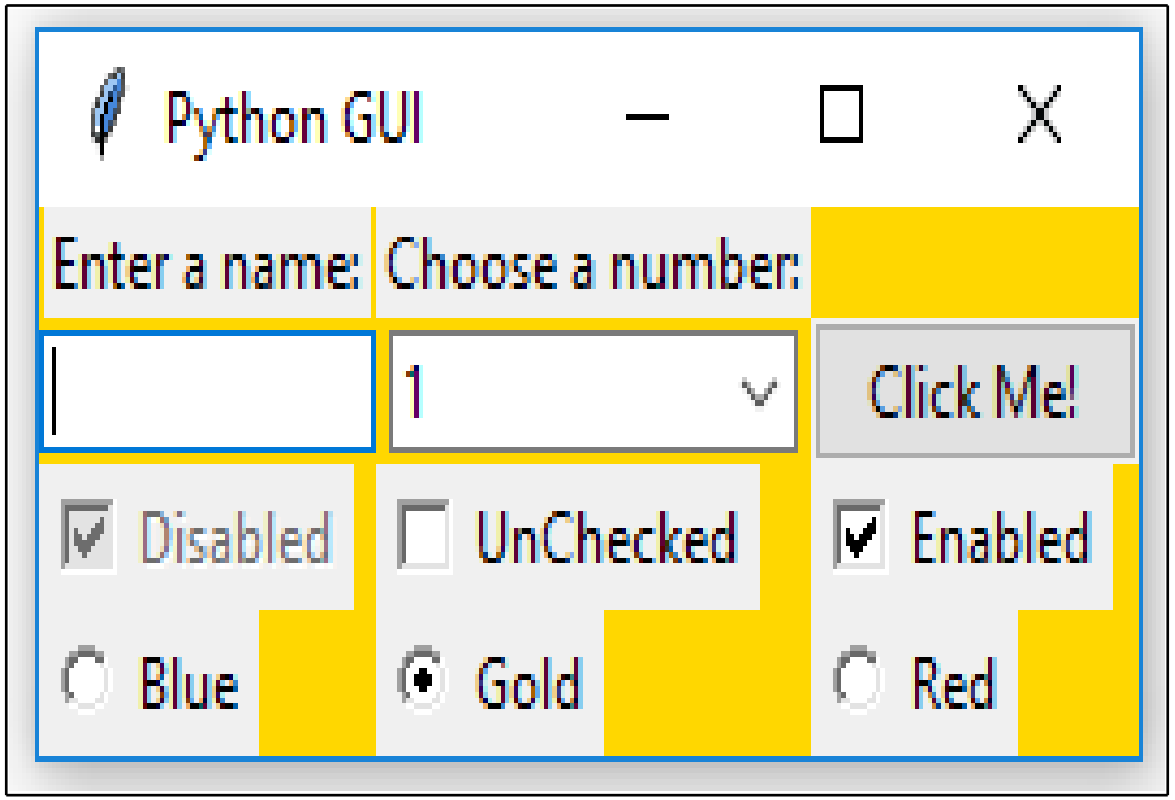
5. Use the grid layout to position them:

```
rad1.grid(column=0, row=5, sticky=tk.W,
columnspan=3)
rad2.grid(column=1, row=5, sticky=tk.W,
columnspan=3)
rad3.grid(column=2, row=5, sticky=tk.W,
columnspan=3)
```

The preceding steps will finally produce the following code (`GUI_radiobutton_widget.py`):

```
74  # Radiobutton Globals
75  COLOR1 = "Blue"
76  COLOR2 = "Gold"
77  COLOR3 = "Red"
78
79  # Radiobutton Callback
80  def radCall():
81      radSel=radVar.get()
82      if   radSel == 1: win.configure(background=COLOR1)
83      elif radSel == 2: win.configure(background=COLOR2)
84      elif radSel == 3: win.configure(background=COLOR3)
85
86  # create three Radiobuttons using one variable
87  radVar = tk.IntVar()
88
89  rad1 = tk.Radiobutton(win, text=COLOR1, variable=radVar, value=1, command=radCall)
90  rad1.grid(column=0, row=5, sticky=tk.W, columnspan=3)
91
92  rad2 = tk.Radiobutton(win, text=COLOR2, variable=radVar, value=2, command=radCall)
93  rad2.grid(column=1, row=5, sticky=tk.W, columnspan=3)
94
95  rad3 = tk.Radiobutton(win, text=COLOR3, variable=radVar, value=3, command=radCall)
96  rad3.grid(column=2, row=5, sticky=tk.W, columnspan=3)
97
98  name_entered.focus()      # Place cursor into name Entry
99  #======================
100 # Start GUI
101 #======================
102 win.mainloop()
```

6. Run the code. Running this code and selecting the radio button named **Gold**
creates the following window:



There's more…

Here is a small sample of the available symbolic color names that you can look up in the official TCL documentation at http://www.tcl.tk/man/tcl8.5/TkCmd/colors.htm:

Name	Red	Green	Blue
`alice blue`	240	248	255
`AliceBlue`	240	248	255
`Blue`	0	0	255
`Gold`	255	215	0
`Red`	255	0	0

Some of the names create the same color, so **`alice blue`** creates the same color as **`AliceBlue`**. In this recipe, we used the symbolic names **`Blue`**, **`Gold`**, and **`Red`**.

:

Using scrolled text widgets

`ScrolledText` widgets are much larger than simple `Entry` widgets and span multiple lines. They are widgets like Notepad and wrap lines, automatically enabling vertical scroll bars when the text gets larger than the height of the `ScrolledText` widget.

:

How to do it…

By adding the following lines of code, we create a `ScrolledText` widget:

1. Start with the `GUI_radiobutton_widget.py` module and save it as `GUI_scrolledtext_widget.py`.

2. Import `scrolledtext`:

```
from tkinter import scrolledtext
```

3. Define variables for the width and height:

```
scrol_w = 30
scrol_h = 3
```

:

How to do it…

By adding the following lines of code, we create a `ScrolledText` widget:

4. Create a `ScrolledText` widget:

```
scr = scrolledtext.ScrolledText(win,
width=scrol_w, height=scrol_h, wrap=tk.WORD)
```

5. Position the widget:

```
scr.grid(column=0, columnspan=3)
```

:

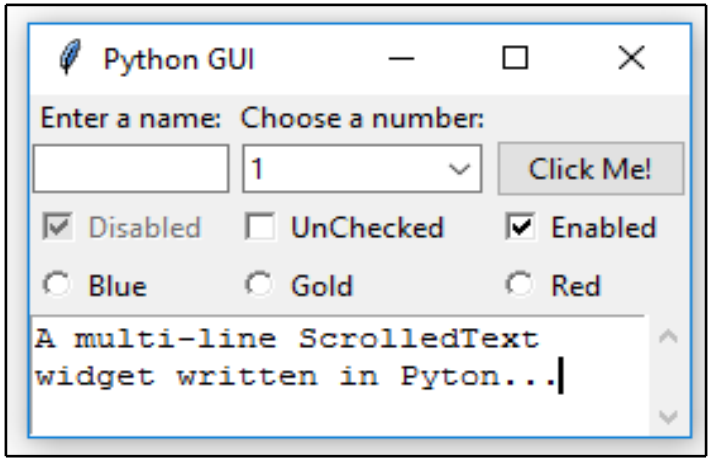
How to do it…

By adding the following lines of code, we create a `ScrolledText` widget:

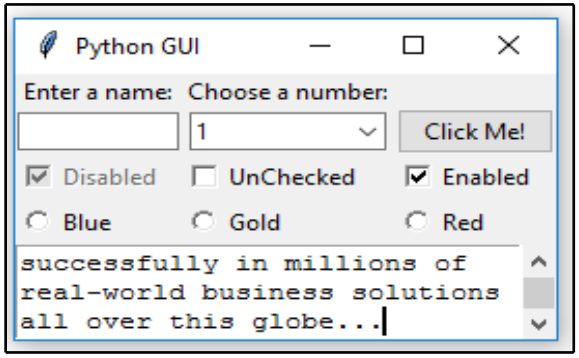
The preceding steps will finally produce the following code (`GUI_scrolledtext_widget.py`):

```
#======================
# imports
#======================
import tkinter as tk
from tkinter import ttk
from tkinter import scrolledtext

# Using a scrolled Text control
scrol_w  = 30
scrol_h  =  3
scr = scrolledtext.ScrolledText(win, width=scrol_w, height=scrol_h, wrap=tk.WORD)
scr.grid(column=0, columnspan=3)

name_entered.focus()      # Place cursor into name Entry
#======================
# Start GUI
#======================
win.mainloop()
```

6. Run the code. We can actually type into our widget, and if we type enough words, the lines will automatically wraparound:



Once we type in more words than the height of the widget can display, the vertical scroll bar becomes enabled. This all works out of the box without us needing to write any more code to achieve this:



Adding several widgets in a loop

So far, we have created several widgets of the same type (for example, a radio button) by basically copying and pasting the same code and then modifying the variations (for example, the column number). In this recipe, we start refactoring our code to make it less redundant.

Getting ready

We are refactoring some parts of the previous recipe's code, *Using scrolled text widgets*, so you need that code for this recipe.

Adding several widgets in a loop

How to do it…

Here's how we refactor our code:

1. Start with the `GUI_scrolledtext_widget.py` module and save it as `GUI_adding_widgets_in_loop.py`.
2. Delete the global name variables and create a Python list instead:

```
colors = ["Blue", "Gold", "Red"]
```

3.Use the `get()` function on the radio button variable:

```
radSel=radVar.get()
```

4.Create logic with an `if ... elif` structure:

```
if radSel == 0:
    win.configure(background=colors[0])
elif radSel == 1:
    win.configure(background=color[1])
elif radSel == 2:
    win.configure(background=color[2])
```

5.Use a loop to create and position the radio buttons:

```
for col in range(3):
  curRad = tk.Radiobutton(win, text=colors[col],
  cariable=radVar, value, command=radCall)
  curRad.brid(column=col, row=5, sticky=tk.W)
```

6.Run the code (`GUI_adding_widgets_in_loop.py`):

```
76  # First, we change our Radiobutton global variables into a list
77  colors = ["Blue", "Gold",  "Red"]
78
79  # We have also changed the callback function to be zero-based, using the list
80  # instead of module-level global variables
81  # Radiobutton Callback
82  def radCall():
83      radSel=radVar.get()
84      if    radSel == 0: win.configure(background=colors[0])  # now zero-based
85      elif radSel == 1: win.configure(background=colors[1])  # and using list
86      elif radSel == 2: win.configure(background=colors[2])
87
88  # create three Radiobuttons using one variable
89  radVar = tk.IntVar()
90
91  # Next we are selecting a non-existing index value for radVar
92  radVar.set(99)
93
94  # Now we are creating all three Radiobutton widgets within one loop
95  for col in range(3):
96      curRad = tk.Radiobutton(win, text=colors[col], variable=radVar,
97                              value=col, command=radCall)
98      curRad.grid(column=col, row=5, sticky=tk.W)
99
```

Running this code will create the same window as before, but our code is much cleaner and easier to maintain. This will help us when we expand our GUI in the coming recipes.