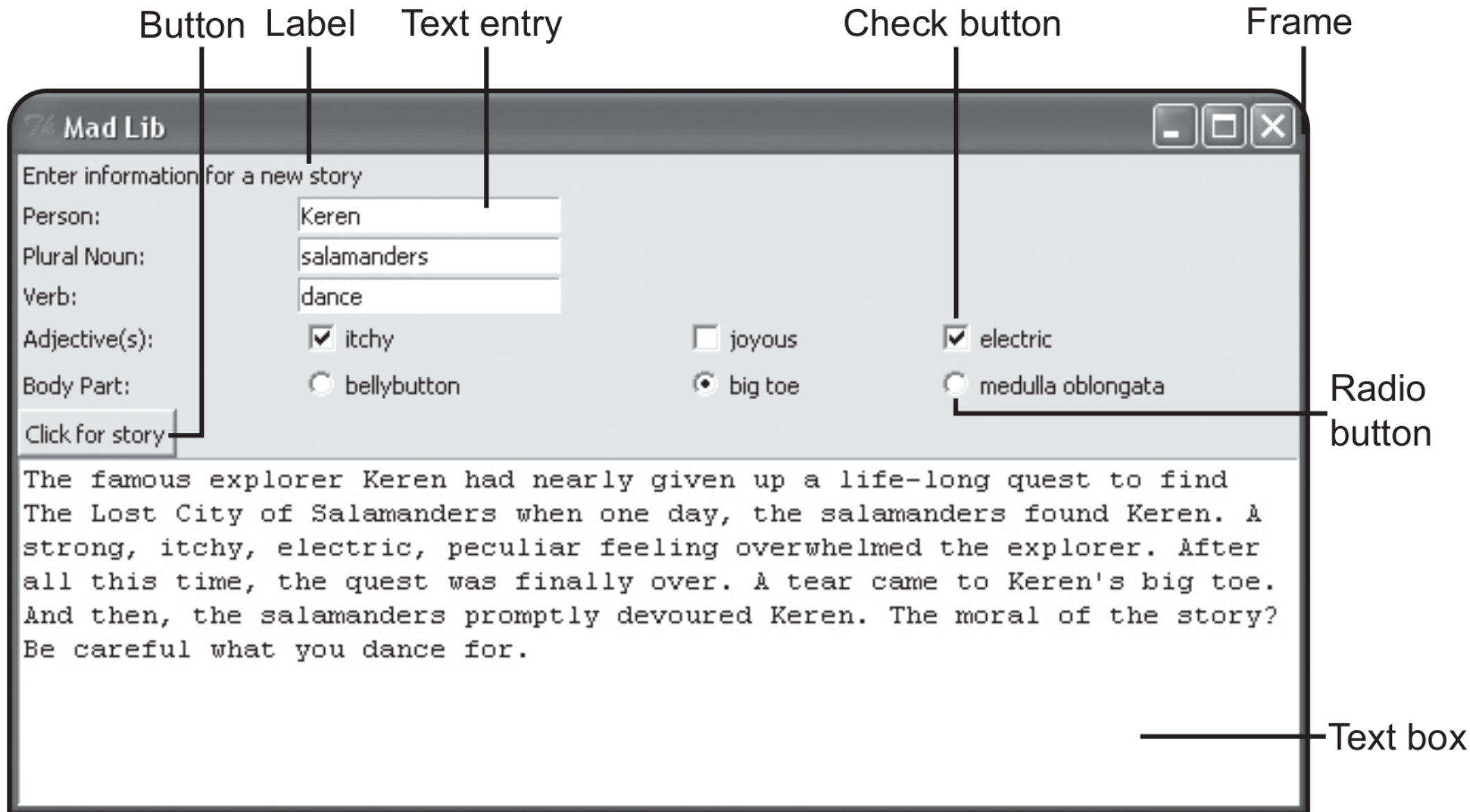


# **Chapter 11**

## **GUI Development: The Mad Lib Program**

# Examining A GUI

- Define all of the GUI (Graphical User Interface) elements you'll meet in this chapter.



- To create a GUI with Python, we need to use a GUI toolkit.
- There are many to pick from, but we use [Tkinter, or TK](#), a popular cross-platform toolkit, here.
- We create GUI elements by instantiating objects from classes of the [tkinter](#) module, part of the [Tkinter](#) toolkit.

Element	tkinter	Class Description
Frame	Frame	Holds other GUI elements
Label	Label	Displays uneditable text or icons
Button	Button	Performs an action when the user activates it
Text entry	Entry	Accepts and displays one line of text
Text box	Text	Accepts and displays multiple lines of text
Check button	Checkbutton	Allows the user to select or not select an option
Radio button	Radiobutton	Allows, as a group, the user to select one option from several

- No need to memorize all of these [tkinter](#) classes.

# Understanding Event-Driven Programming

- GUI programs are *event-driven*, meaning they respond to actions regardless of the order in which they occur.
- For an event-driven program, you *bind* (associate) *events* (things that can happen involving the program's objects) with *event handlers* (code that runs when the events occur).
- By defining all of your objects, events, and event handlers, you establish how your program works. Then, you kick off the program by entering an event loop, where the program waits for the events to occur. When any of those events do occur, the program handles them, just as you've laid out.

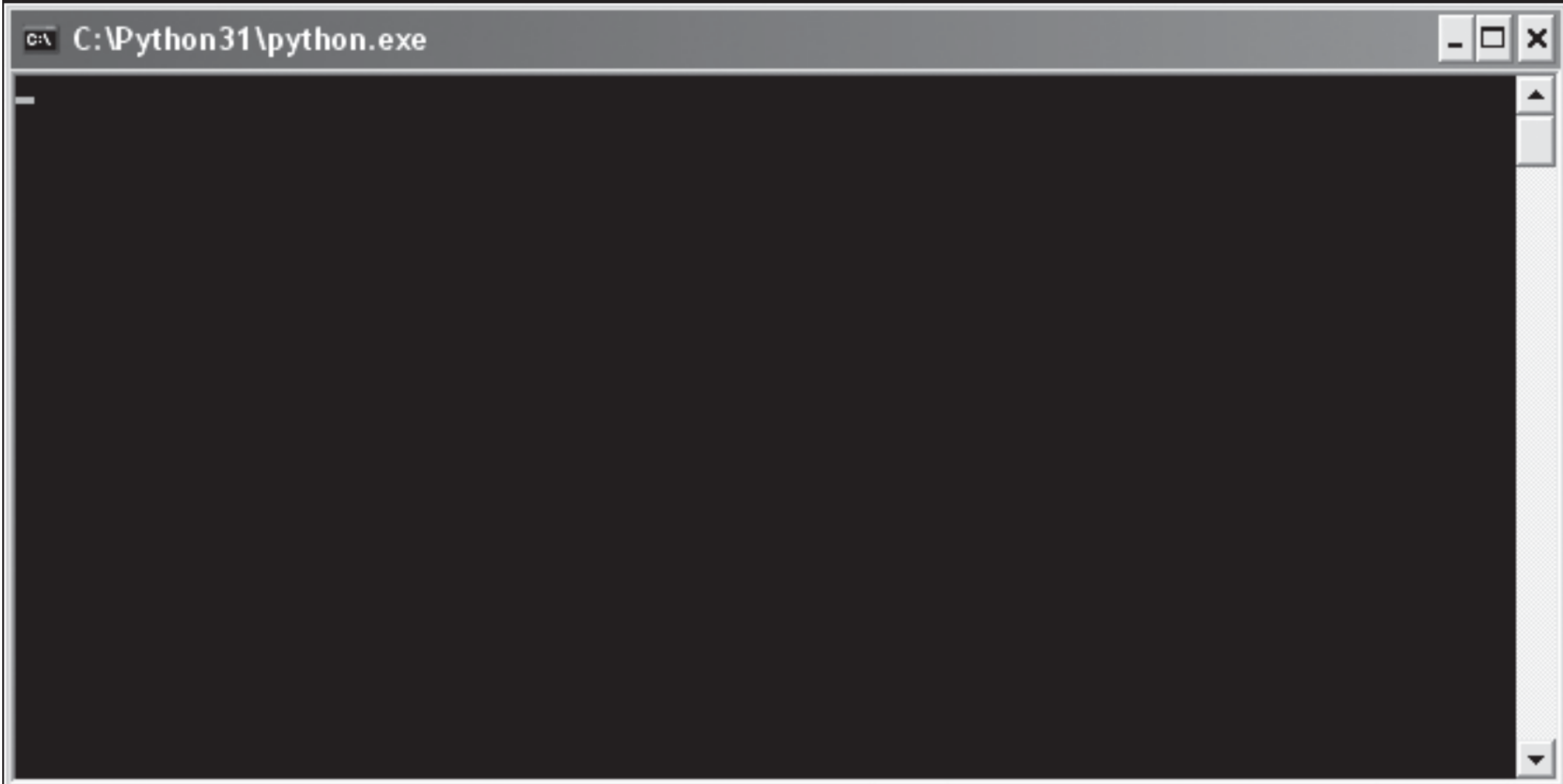
# Introducing the Simple GUI Program

- The batch file: `simple_gui.bat`

`simple_gui.py`

`pause`





- In addition to the window just showed, Simple GUI may generate another window (depending upon your system): the familiar console window.
- On a Windows machine, the easiest way to suppress the accompanying console window is to change the extension of your program from `py` to `pyw`.

# simple\_gui.py

```
# Simple GUI  
# Demonstrates creating a window
```

```
from tkinter import *
```

```
# create the root window  
root = Tk()
```

```
# modify the window  
root.title("Simple GUI")  
root.geometry("200x100")
```

```
# kick off the window's event-loop  
root.mainloop()
```

# Importing the tkinter Module

```
from tkinter import *
```

- The code imports all of `tkinter` directly into the program's global scope.
- Normally, you want to avoid doing something like this; however, `tkinter` is designed to be imported in this way.



# Creating a Root Window

- To create a root window, we instantiate an object of the `tkinter` class `Tk`:

```
root = Tk()
```

- Notice that we didn't have to prefix the module name, `tkinter`, to the class name, `Tk`.
- In fact, we can now directly access any part of the `tkinter` module, without having to use the module name. This saves a lot of typing and makes code easier to read.
- You can have only one root window in a `Tkinter` program.

# Modifying a Root Window

- Modify the root window using a few of its methods:

```
root.title("Simple GUI")  
root.geometry("200x100")
```

- **title()** sets the title of the root window. All you have to do is pass the title you want displayed as a string.
- **geometry()** sets the size of the root window, in pixels. The method takes a **string** (and not integers) that represents the window's width and height, separated by the "x" character.

# Entering a Root Window's Event Loop

- Start up the window's event loop by invoking `root`'s **`mainloop()`**:

**`root.mainloop()`**

- As a result, the window stays open, waiting to handle events.

# Using Labels

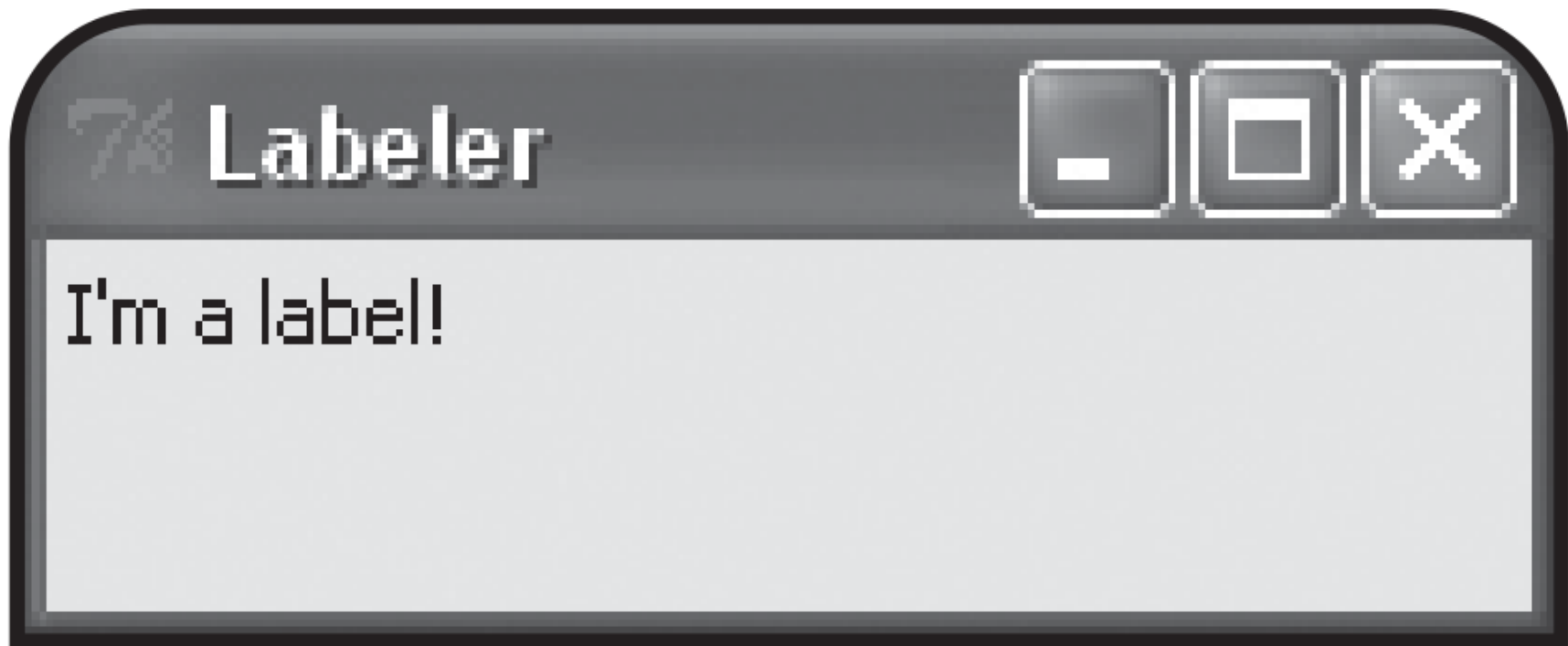
- GUI elements are called *widgets*, short for window gadgets.
- The simplest widget is the **Label** widget, which is uneditable text and/or icons.
- A **Label** widget labels part of a GUI. It's often used to label other widgets. And labels aren't interactive.

# Introducing the Labeler Program

- The batch file: [labeler.bat](#)

**labeler.py**

**pause**



```
# Labeler  
# Demonstrates a label
```

```
from tkinter import *
```

```
# create the root window  
root = Tk()  
root.title("Labeler")  
root.geometry("200x50")
```

```
# create a frame in the window to hold other widgets  
app = Frame(root)  
app.grid()
```

```
# create a label in the frame  
lbl = Label(app, text = "I'm a label!")  
lbl.grid()
```

```
# kick off the window's event loop  
root.mainloop()
```

**labeler.py**

# Creating a Frame

- A **Frame** is a widget that can hold other widgets (eg, **Label** widget). You use it as a base on which to place other things:

```
app = Frame(root)
```

- When you create a new widget, you must pass its *master* (the thing that will contain the widget) to the constructor of the new object. Here, we pass **root** to the **Frame** constructor.
- Invoke the **grid()** method of the new object:

```
app.grid()
```

- **grid()** is a method that all widgets have. It's associated with a *layout manager*, which lets you arrange widgets.

# Creating a Label

- create a **Label** widget by instantiating an object of the **Label** class:

```
lbl = Label(app, text = "I'm a label!")
```

- By passing `app` to the **Label** object's constructor, we make the frame that `app` refers to the master of the **Label** widget. As a result, the label is placed in the frame.
- By passing `"I'm a label!"` to the `text` parameter, we set the widget's `text` option to that string.
- Invoke the object's **grid()** method:

```
lbl.grid()
```

ensures that the label will be visible.

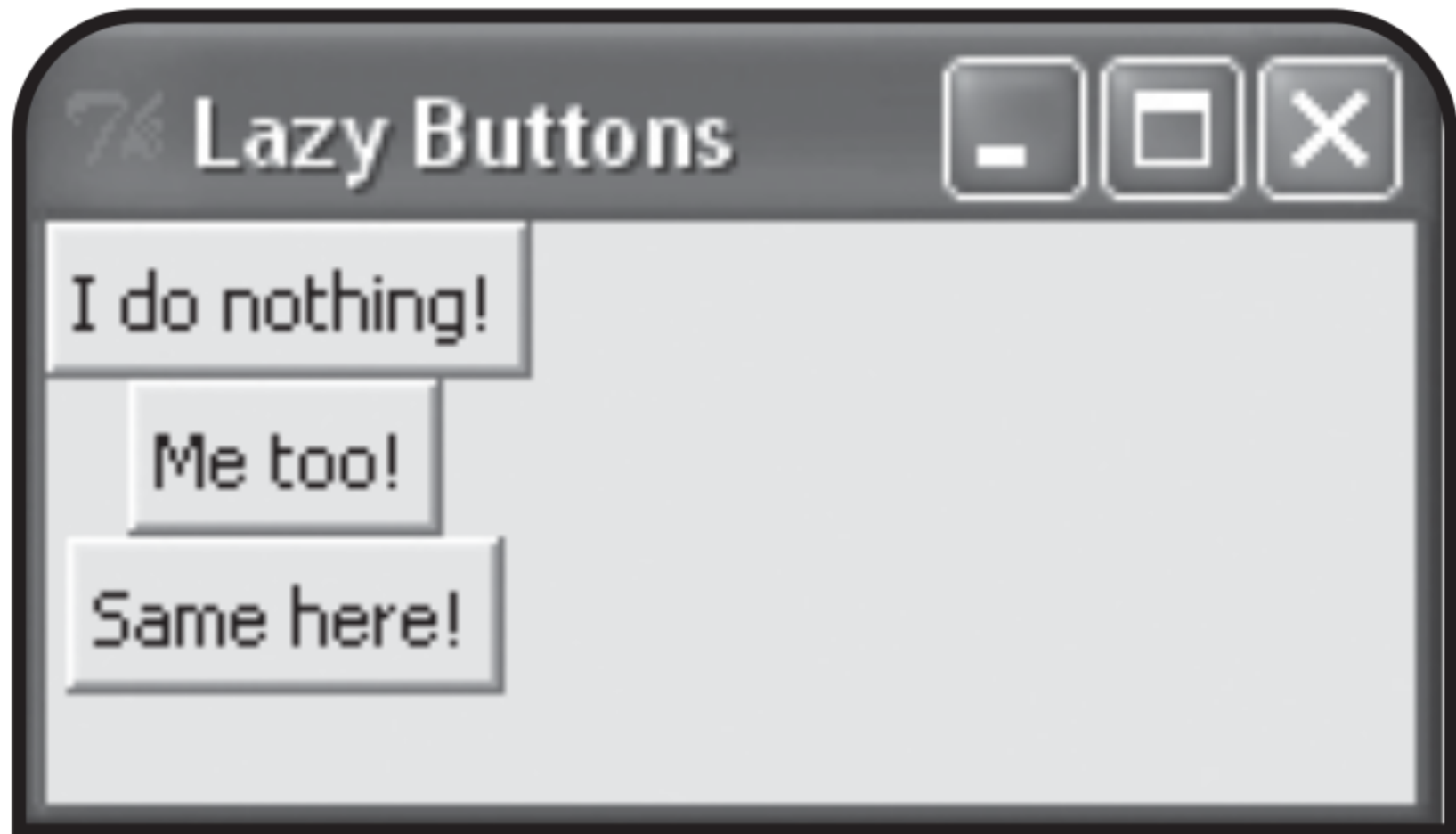


# Introducing the Lazy Buttons Program

- The batch file: [lazy.buttons.bat](#)

[lazy\\_buttons.py](#)

**pause**



# lazy\_buttons.py

```
# Lazy Buttons  
# Demonstrates creating buttons
```

```
from tkinter import *
```

```
# create a root window
```

```
root = Tk()
```

```
root.title("Lazy Buttons")
```

```
root.geometry("200x85")
```

```
# create a frame in the window to hold other widgets
```

```
app = Frame(root)
```

```
app.grid()
```

```
# create a button in the frame
```

```
btn1 = Button(app, text = "I do nothing!")
```

```
btn1.grid()
```

```
# create a second button in the frame  
btn2 = Button(app)  
btn2.grid()  
btn2.configure(text = "Me too!")
```

```
# create a third button in the frame  
btn3 = Button(app)  
btn3.grid()  
btn3["text"] = "Same here!"
```

```
# kick off the root window's event loop  
root.mainloop()
```

# Creating Buttons

- Create a **Button** widget by instantiating an object of the **Button** class:

```
btn1 = Button(app, text = "I do nothing!")  
btn1.grid()
```

- These lines create a new button with **I do nothing!** The button's master is the frame we created earlier, which means that the button is placed in the frame.
- You can create a widget and set all of its options in one line, or you can create a widget and set or alter its options later:

```
btn2 = Button(app)  
btn2.grid()
```

- The only value we pass to the object's constructor is **app**, so all we have done is add a blank button to the frame.

- We can modify a widget after we create it, using the object's **configure()** method:

```
btn2.configure(text = "Me too!")
```

- You can use a widget's **configure()** for any widget option (and any type of widget). You can even use the method to change an option that you've already set.

- Create a 3<sup>rd</sup> button:

```
btn3 = Button(app)  
btn3.grid()
```

- Set the button's **text** option, using a different interface:

```
btn3["text"] = "Same here!"
```

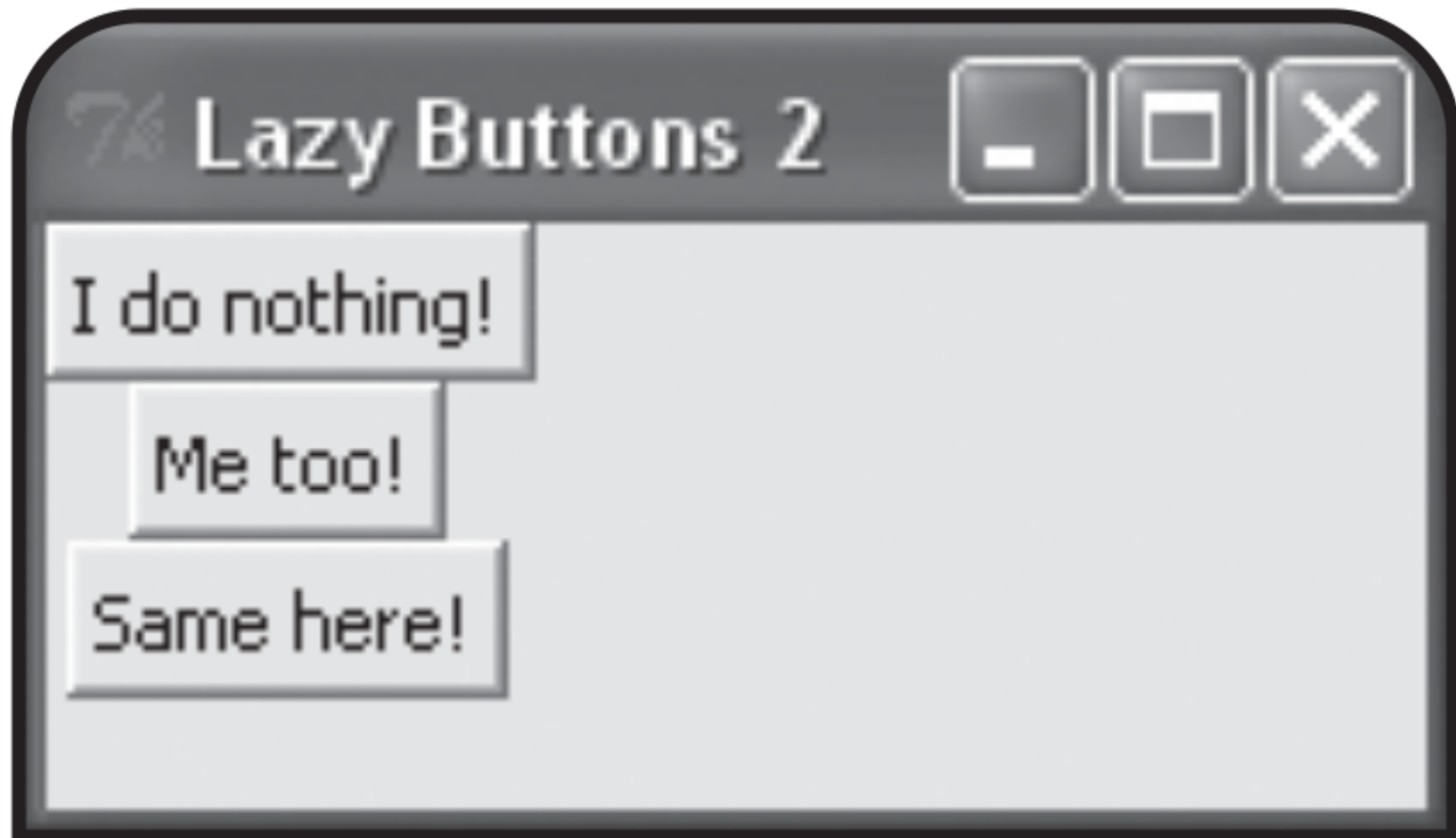
- We access the button's **text** option through a dictionary-like interface. I set the **text** option to **"Same here!"** , which puts the text **Same here!** on the button.

# Introducing the Lazy Buttons 2 Program

- The batch file: [lazy.buttons2.bat](#)

[lazy\\_buttons2.py](#)

**pause**



# lazy\_buttons2.py

```
# Lazy Buttons 2
```

```
# Demonstrates using a class with Tkinter
```

```
from tkinter import *
```

```
class Application(Frame):
```

```
    """ A GUI application with three buttons. """
```

```
    def __init__(self, master):
```

```
        """ Initialize the Frame. """
```

```
        super(Application, self).__init__(master)
```

```
        self.grid()
```

```
        self.create_widgets()
```

```
    def create_widgets(self):
```

```
        """ Create three buttons that do nothing. """
```

```
        # create first button
```

```
        self.btnn1 = Button(self, text = "I do nothing!")
```

```
        self.btnn1.grid()
```

```
# create second button
self.btn2 = Button(self)
self.btn2.grid()
self.btn2.configure(text = "Me too!")
```

```
# create third button
self.btn3 = Button(self)
self.btn3.grid()
self.btn3["text"] = "Same here!"
```

```
# main
root = Tk()
root.title("Lazy Buttons 2")
root.geometry("200x85")
app = Application(root)
root.mainloop()
```



# Defining the Application Class

- Create a new class, `Application`, based on `Frame`:

```
class Application(Frame):
```

```
    """ A GUI application with three buttons. """
```

- Instead of instantiating a `Frame` object, we'll end up instantiating an `Application` object to hold all of the buttons.

# Defining a Constructor Method

- Define `Application`'s constructor:

```
def __init__(self, master):  
    """ Initialize the Frame. """  
    super(Application, self).__init__(master)  
    self.grid()  
    self.create_widgets()
```

- The 1<sup>st</sup> thing to do is call the superclass constructor. We pass along the `Application` object's `master`, so it gets set as the `master`.
- Finally, we invoke the `Application` object's `create_widgets()`, defined next.

# Defining a Method to Create the Widgets

- `create_widgets()` creates all three buttons:

```
def create_widgets(self):
```

```
    # create first button
```

```
    self.btnn1 = Button(self, text = "I do nothing!")
```

```
    self.btnn1.grid()
```

```
    self.btnn2 = Button(self) # create 2nd button
```

```
    self.btnn2.grid()
```

```
    self.btnn2.configure(text = "Me too!")
```

```
    self.btnn3 = Button(self) # create 3rd button
```

```
    self.btnn3.grid()
```

```
    self.btnn3["text"] = "Same here!"
```

- Here `btnn1`, `btnn2`, `btnn3` are attributes of an `Application` object. And we use `self` as the master for the buttons so that the `Application` object is their master.

# Creating the Application Object

- In the main section of code, we create a root window:

```
root = Tk()  
root.title("Lazy Buttons 2")  
root.geometry("200x85")
```

- Then instantiate an `Application` object with the root window as its master:

```
app = Application(root)
```

- The `Application` object's constructor invokes the object's `create_widgets()`. This method then creates the 3 buttons, with the `Application` object as their master.

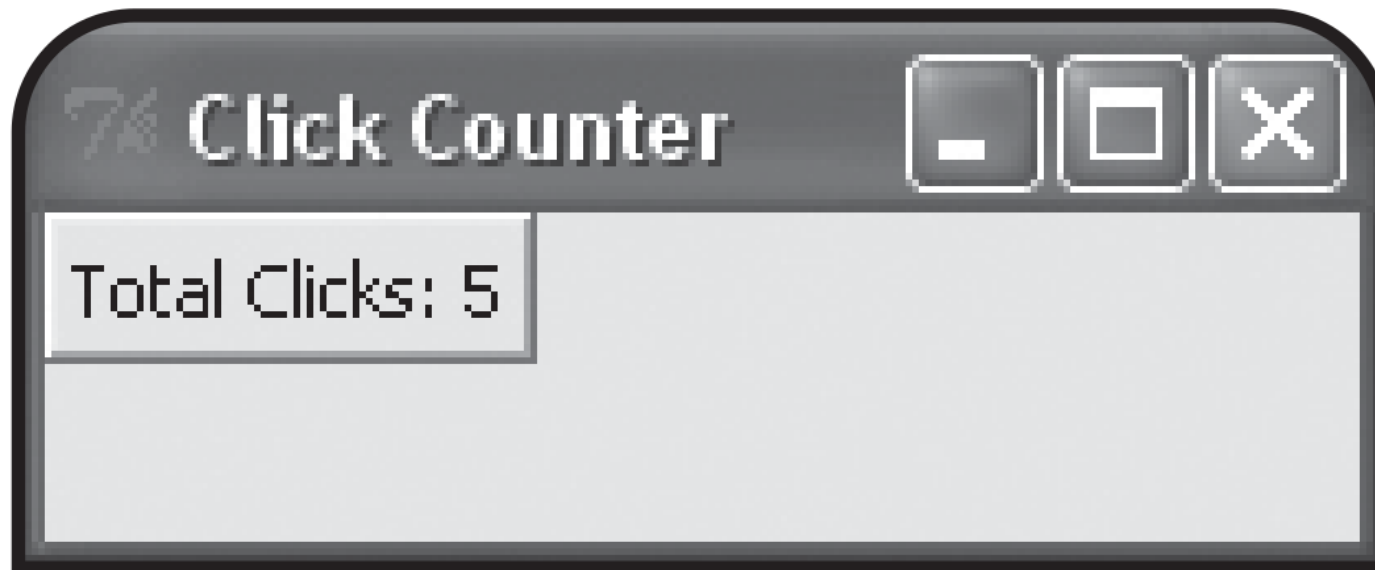
- Finally, we invoke the root window's event loop:

```
root.mainloop()
```

# Introducing the Click Counter Program

- The batch file: [click\\_counter.bat](#)

[click\\_counter.py](#)  
**pause**



# click\_counter.py

```
# Click Counter
```

```
# Demonstrates binding events with an event handler
```

```
from tkinter import *
```

```
class Application(Frame):
```

```
    """ GUI application which counts button clicks. """
```

```
    def __init__(self, master):
```

```
        """ Initialize the frame. """
```

```
        super(Application, self).__init__(master)
```

```
        self.grid()
```

```
        # the number of button clicks
```

```
        self.btn_clicks = 0
```

```
        self.create_widget()
```

```
def create_widget(self):  
    """ Create button displaying number of clicks. """  
    self.btttn = Button(self)  
    self.btttn["text"] = "Total Clicks: 0"  
    self.btttn["command"] = self.update_count  
    self.btttn.grid()
```

```
def update_count(self):  
    """ Increase click count and display new total. """  
    self.btttn_clicks += 1  
    self.btttn["text"] = "Total Clicks: " \  
        + str(self.btttn_clicks)
```

```
# main  
root = Tk()  
root.title("Click Counter")  
root.geometry("200x50")
```

```
app = Application(root)
```

```
root.mainloop()
```

# Setting Up the Program

- start the `Application` class definition:

```
class Application(Frame):  
    """ GUI application which counts button clicks. """  
    def __init__(self, master):  
        """ Initialize the frame. """  
        super(Application, self).__init__(master)  
        self.grid()  
        self.bttm_clicks = 0 # the number of button clicks  
        self.create_widget()
```

- We create an object attribute `self.bttm_clicks` to keep track of the number of times the user clicks the button.



# Binding the Event Handler

- In the `create_widget()` method, we create a single button:

```
def create_widget(self):  
    """ Create button displaying number of clicks. """  
    self.bbtn = Button(self)  
    self.bbtn["text"] = "Total Clicks: 0"  
    self.bbtn["command"] = self.update_count  
    self.bbtn.grid()
```

- We set the `Button` widget's **command** option to the `update_count()` method. So when the user clicks the button, the method is invoked.
- Technically, what we've done is bind an event (the clicking of `Button` widget) to an event handler (ie, `update_count()`).
- In general, you set a widget's **command** option to bind the activation of the widget with an event handler.

# Creating the Event Handler

- `update_count()` handles the event of the button being clicked:

```
def update_count(self):  
    """ Increase click count and display new total. """  
    self.btt_n_clicks += 1  
    self.btt_n["text"] = "Total Clicks: " \  
        + str(self.btt_n_clicks)
```

- This method increments the total number of button clicks and then changes the text of the button to reflect the new total.

# Wrapping Up the Program

- The main part of the code:

```
# main
root = Tk()
root.title("Click Counter")
root.geometry("200x50")

app = Application(root)

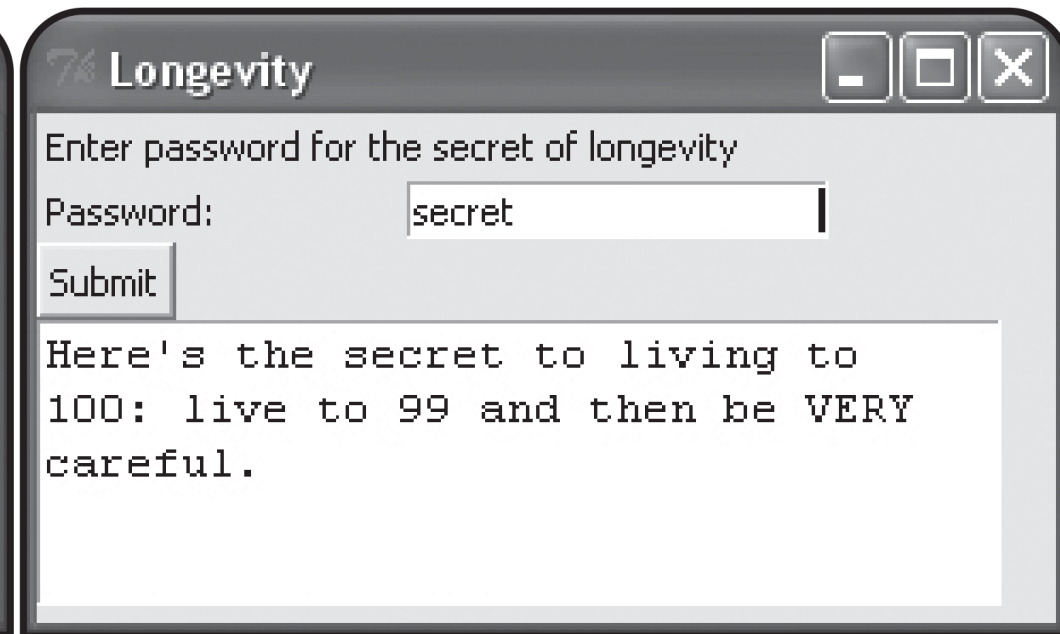
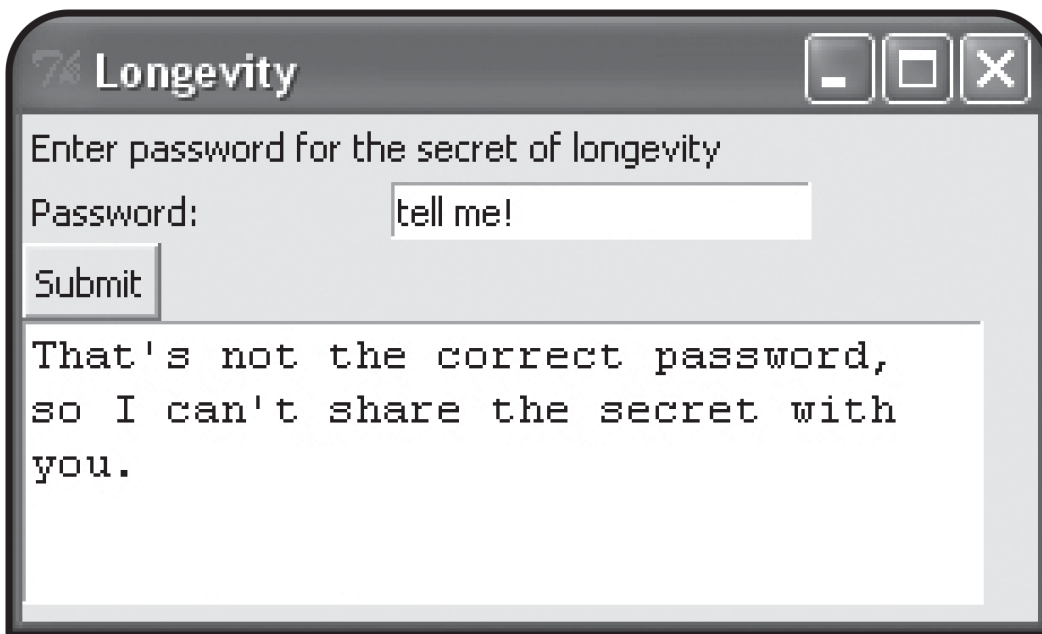
root.mainloop()
```

- We create a root window and set its title and dimensions. Then I instantiate a new [Application](#) object with the root window as its master. Lastly, I start up the root window's event loop to bring the GUI to life on the screen.

# Introducing the Longevity Program

- The batch file: [longevity.bat](#)

**longevity.py**  
**pause**



# longevity.py

```
# Longevity
# Demonstrates text, entry widgets, grid layout manager

from tkinter import *

class Application(Frame):
    """ GUI application reveals the secret of longevity. """
    def __init__(self, master):
        super(Application, self).__init__(master)
        self.grid()
        self.create_widgets()

    def create_widgets(self):
        # create instruction label
        self.inst_lbl = Label(self, text="Enter password" +\
                               " for the secret of longevity")
        self.inst_lbl.grid(row = 0, column = 0,
                           columnspan = 2, sticky = W)
```



```
def reveal(self):  
    """ Display message based on password. """  
    contents = self.pw_ent.get()  
    if contents == "secret":  
        message="Here's the secret to living to 100:"+\  
            " live to 99 and then be VERY careful."  
    else:  
        message = "That's not the correct password,"+\  
            "so I can't share the secret with you."  
    self.secret_txt.delete(0.0, END)  
    self.secret_txt.insert(0.0, message)
```

```
# main
```

```
root = Tk()
```

```
root.title("Longevity")
```

```
root.geometry("300x150")
```

```
app = Application(root)
```

```
root.mainloop()
```

# Placing a Widget with the Grid Layout Manager

- Start `create_widgets()` and create a label that provides instructions to the user:

```
def create_widgets(self):  
    self.inst_lbl=Label(self, text="Enter password"+\  
        “ for the secret of longevity”)
```

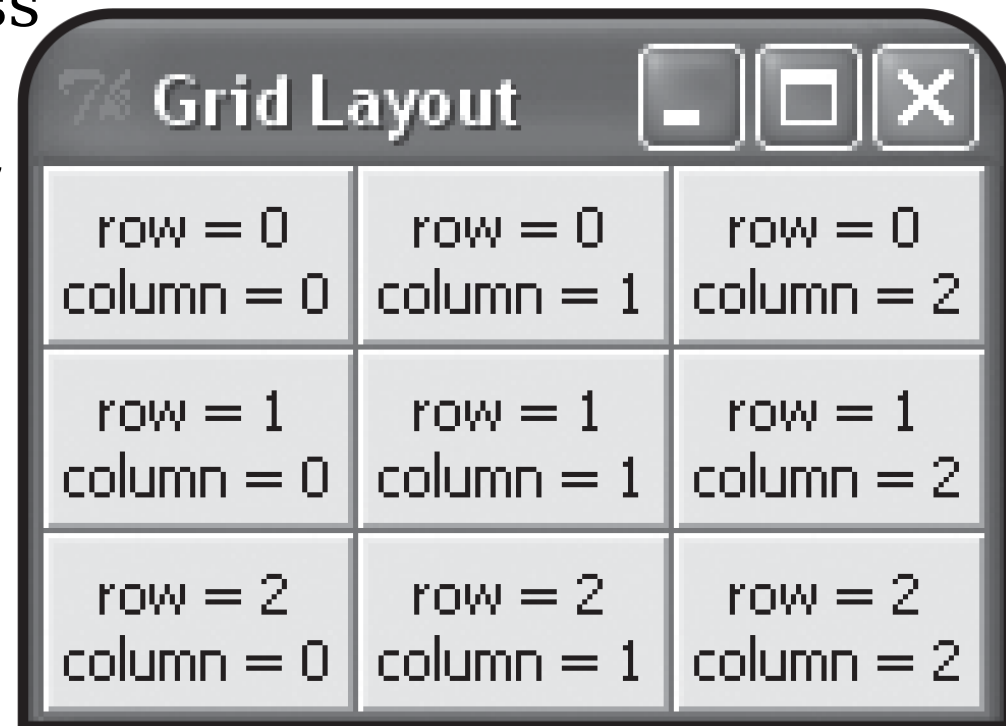
- Use the `grid` layout manager to be specific about the placement of this label:

```
self.inst_lbl.grid(row = 0, column = 0,  
    columnspan = 2, sticky = W)
```

- A widget object's `grid()` method can take values for many different parameters, but we only use 4 of them: **row**, **column**, **columnspan**, and **sticky**.



- The **row** and **column** parameters take integers and define where an object is placed within its master widget.
- Imagine the frame in the root window as a grid, divided into rows and columns. At each row and column intersection is a cell, where you can place a widget.
- For our **Label** widget, we pass 0 to **row**, 0 to **column**, which puts the label in the upper-left corner of the frame.
- If a widget is wide, you may want to allow the widget to span more than one cell so that your other widgets are correctly spaced.



- **columnspan** lets you span a widget over more than one column. We pass 2 to this parameter to allow the long label to span 2 columns.

- So the label takes up 2 cells, the one at row 0, column 0, and the other at row 0, column 1.
- You can also use the **rowspan** parameter to allow a widget to span more than one row.
- You can justify the widget within the cell by using **sticky**, which takes directions as values, including **N**, **S**, **E**, and **W**.
- Since we pass **W** to **sticky** for the **Label** object, the label is forced to the west (left).
- Create a label that appears in the next row, left-justified:

```
self.pw_lbl = Label(self, text = "Password: ")  
self.pw_lbl.grid(row = 1, column = 0, sticky = W)
```

# Creating an Entry Widget

- Create a new type of widget, an **Entry** widget:

```
self.pw_ent = Entry(self)  
self.pw_ent.grid(row = 1, column = 1, sticky = W)
```

creates a text entry where the user can enter a password, and position the **Entry** in the cell next to the password label.

- Create a button to submit the password:

```
self.submit_btn = Button(self, text = "Submit",  
                           command = self.reveal)  
self.submit_btn.grid(row=2,column=0, sticky=W)
```

bind the activation of the button with `reveal()`, which reveals the longevity secret, if the user has entered the correct password. And place the button in the next row, all the way to the left.

# Creating a Text Widget

- Create a new type of widget, a **Text** widget:

```
self.secret_txt = Text(self, width = 35, height = 5,  
                        wrap = WORD)  
self.secret_txt.grid(row = 3, column = 0,  
                     colspan = 2, sticky = W)
```

- We pass values to **width** & **height** to set the text box. Then we pass a value to the parameter **wrap**, which could be
  - \* **WORD** wraps entire words when you reach the right edge of the text box.
  - \* **CHAR** wraps character, meaning that when you get to the right edge of the text box, the next character simply appears on the following line.
  - \* **NONE** means no wrapping. As a result, you can only write text on the 1<sup>st</sup> line of the text box.

# Getting and Inserting Text with Text-Based Widgets

- `reveal()` tests if the user has entered the correct password. If so, the method displays the secret to a long life. Otherwise the user is told that the password is incorrect.

- Firstly get the text in the `Entry` widget by invoking its `get()`

```
def reveal(self):  
    contents = self.pw_ent.get()
```

- `get()` returns the text in the widget. Both `Entry` and `Text` objects have a `get()` method.

- Check if the text is equal to `"secret"`. If so, set `message` to the string describing the secret to living to 100. Otherwise, set `message` to the string that tells the user that he entered the wrong password:

```
if contents == "secret":
    message="Here's the secret to living to 100:"+\
           " live to 99 and then be VERY careful."
else:
    message = "That's not the correct password,"+\
             "so I can't share the secret with you."
```

- Now we have the string that we want to show to the user, we need to insert it into the `Text` widget.
- 
- Firstly, delete any text already in the `Text` widget by invoking its `delete()`:

```
self.secret_txt.delete(0.0, END)
```

- `delete()` can delete text from text-based widgets. It can take a single index, or a beginning and an ending point.

- Pass floating-point numbers to represent a row and column number pair where the digit to the left of the decimal point is the row number and the digit to the right of the decimal point is the column number.
- Pass 0.0 as the starting point, meaning that the method should delete text starting at row 0, column 0 (the absolute beginning) of the text box.
- **END** means the end of the text. So, this line of code deletes everything from the 1<sup>st</sup> position in the text box to the end. Both **Text** and **Entry** widgets have a **delete()** method.
- Insert the string we want to display into the **Text** widget:

```
self.secret_txt.insert(0.0, message)
```

- **insert()** inserts a string into a text-based widget. The method takes an insertion position and a string.

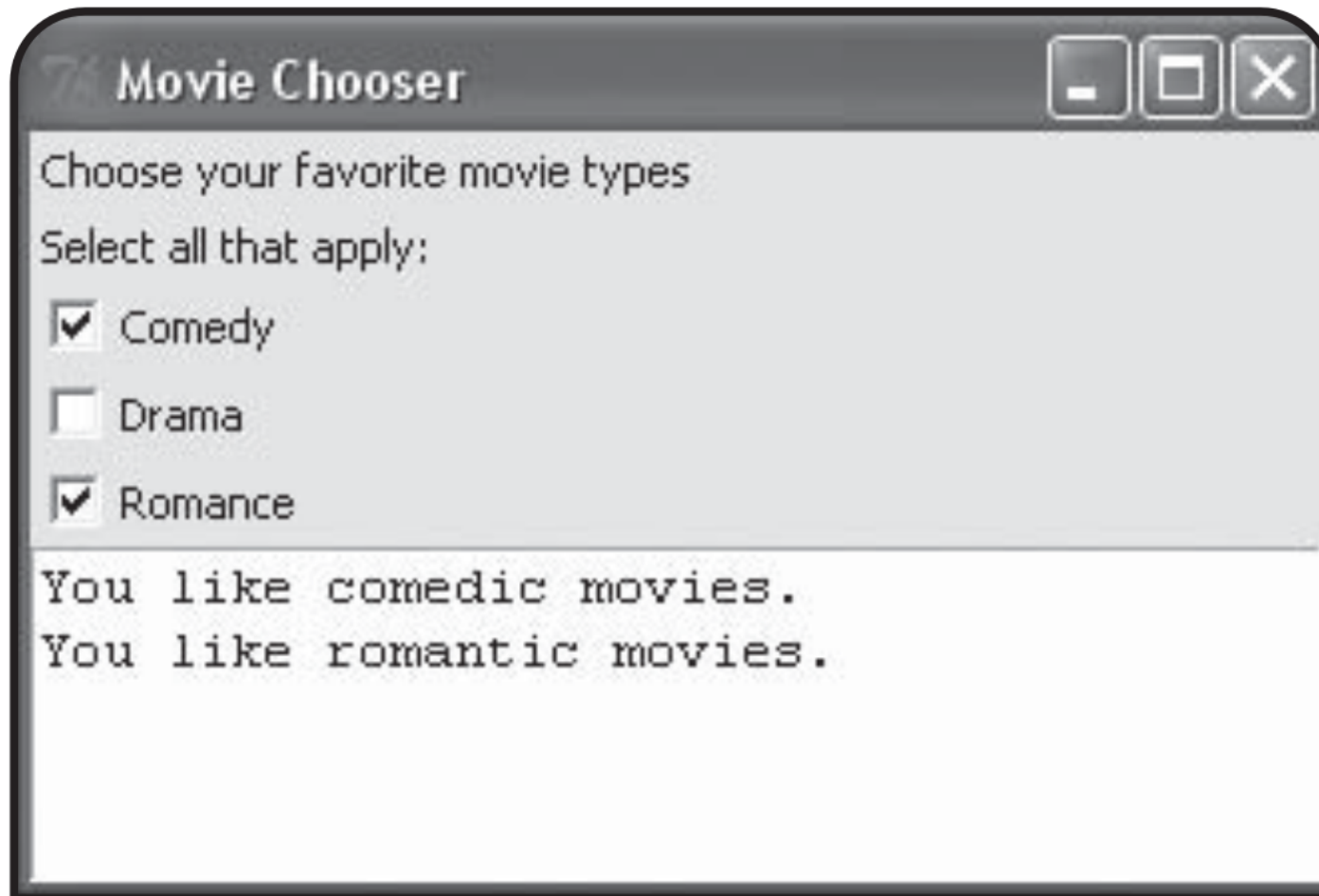
- We pass 0.0 as the insertion position, meaning the method should start inserting at row 0, column 0. We pass `message` as the 2<sup>nd</sup> value, so that the appropriate message shows up in the text box.
- Both `Text` and `Entry` widgets have an `insert()` method.
- `insert()` doesn't replace the text in a text-based widget—it simply inserts it. If you want to replace the existing text with new text, first call the text-based widget's `delete()` method.



# Introducing the Movie Chooser Program

- The batch file: [movie\\_chooser.bat](#)

**movie\_chooser.py**  
**pause**



# movie\_chooser.py

```
# Movie Chooser  
# Demonstrates check buttons
```

```
from tkinter import *
```

```
class Application(Frame):
```

```
    """ GUI Application for favorite movie types. """
```

```
    def __init__(self, master):
```

```
        super(Application, self).__init__(master)
```

```
        self.grid()
```

```
        self.create_widgets()
```

```
    def create_widgets(self):
```

```
        """ Create widgets for movie type choices. """
```

```
        # create description label
```

```
        Label(self,
```

```
            text = "Choose your favorite movie types"\  
        ).grid(row = 0, column = 0, sticky = W)
```

```
# create instruction label
Label(self, text = "Select all that apply:"\
      ).grid(row = 1, column = 0, sticky = W)

# create Comedy check button
self.likes_comedy = BooleanVar()
Checkbutton(self, text = "Comedy",
             variable = self.likes_comedy,
             command = self.update_text\
             ).grid(row = 2, column = 0, sticky = W)

# create Drama check button
self.likes_drama = BooleanVar()
Checkbutton(self, text = "Drama",
             variable = self.likes_drama,
             command = self.update_text \
             ).grid(row = 3, column = 0, sticky = W)
```

```
# create Romance check button
self.likes_romance = BooleanVar()
Checkbutton(self, text = "Romance",
            variable = self.likes_romance,
            command = self.update_text\
            ).grid(row = 4, column = 0, sticky = W)
```

```
# create text field to display results
self.results_txt = Text(self, width=40, height = 5,
                        wrap = WORD)
self.results_txt.grid(row=5, column=0,
                      colspan=3)
```

```
def update_text(self):
    """ Update and display user's favorite movie. """
    likes = ""

    if self.likes_comedy.get():
        likes += "You like comedic movies.\n"
```

```
if self.likes_drama.get():  
    likes += "You like dramatic movies.\n"
```

```
if self.likes_romance.get():  
    likes += "You like romantic movies."
```

```
self.results_txt.delete(0.0, END)  
self.results_txt.insert(0.0, likes)
```

```
# main
```

```
root = Tk()  
root.title("Movie Chooser")  
app = Application(root)  
root.mainloop()
```

# Allowing a Widget's Master to Be Its Only Reference

- Create a label that describes the program:

```
def create_widgets(self):
```

```
    Label(self, text="Choose your favorite movie types"\  
          ).grid(row = 0, column = 0, sticky = W)
```

- The important difference between this label and the earlier: we don't assign the resulting Label object to a variable. This would usually be a mistake, rendering the object useless because it isn't connected to the program in any way.
- With `tkinter`, a `Label` object is connected to the program, like all GUI elements, by its master.
- So if we won't need to directly access a widget, we don't need to assign the object to a variable. The main benefit of this approach is shorter, cleaner code.

- So the master of the `Label` object is the only reference to it.
- Create another label in much the same way:

```
# create instruction label
Label(self, text = "Select all that apply:"\
      ).grid(row = 1, column = 0, sticky = W)
```

to provides instructions, telling the user that he can select as many movie types as apply.

# Creating Check Buttons

- Create the check buttons, one for each movie type.
- Every check button needs a special object associated with it that automatically reflects the check button's status.
- The special object must be an instance of the **BooleanVar** class from the `tkinter` module. A Boolean *variable* is a special kind of variable that can be only `True` or `False`.
- We instantiate a `BooleanVar` object and assign it to a new object attribute, `likes_comedy`, before we create the Comedy check button:

```
self.likes_comedy = BooleanVar()
```

- Next, we create the check button itself:



```
Checkbutton(self, text = "Comedy",  
             variable = self.likes_comedy,  
             command = self.update_text\  
             ).grid(row = 2, column = 0, sticky = W)
```

creates a new check button with the text `Comedy`.

- By passing `self.likes_comedy` to **variable**, we associate the check button's status (selected or unchecked) with the `likes_comedy` attribute.
- By passing `self.update_text()` to **command**, we bind the activation of the check button with `update_text()`. Whenever the user selects or clears the check button, the `update_text()` method is invoked.
- We don't assign the resulting **Checkbutton** object to a variable because what we care about is the status of the button, which I can access from the `likes_comedy` attribute.

- Create the next 2 check buttons in the same way:

```
self.likes_drama = BooleanVar()  
Checkbutton(self, text = "Drama",  
variable = self.likes_drama,  
command = self.update_text\  
 ).grid(row = 3, column = 0, sticky = W)
```

```
self.likes_romance = BooleanVar()  
Checkbutton(self, text = "Romance",  
variable = self.likes_romance,  
command = self.update_text\  
 ).grid(row = 4, column = 0, sticky = W)
```

- Whenever the user selects or clears the Drama or Romance check buttons, `update_text()` is invoked.

- Even though we don't assign the resulting **Checkbutton** objects to any variables, we can always see the status of the **Drama/Romance** check button through the `likes_drama/likes_romance` attribute.

- Finally, we create the text box to show the results of the user's selections:

```
# create text field to display results
self.results_txt = Text(self, width = 40, height = 5,
                        wrap = WORD)
self.results_txt.grid(row=5,column=0,columnspan=3)
```

# Getting the Status of a Check Button

- `update_text()` updates the text box to reflect the check buttons the user has selected:

```
def update_text(self):  
    likes = ""  
  
    if self.likes_comedy.get():  
        likes += "You like comedic movies.\n"  
    if self.likes_drama.get():  
        likes += "You like dramatic movies.\n"  
    if self.likes_romance.get():  
        likes += "You like romantic movies."  
  
    self.results_txt.delete(0.0, END)  
    self.results_txt.insert(0.0, likes)
```

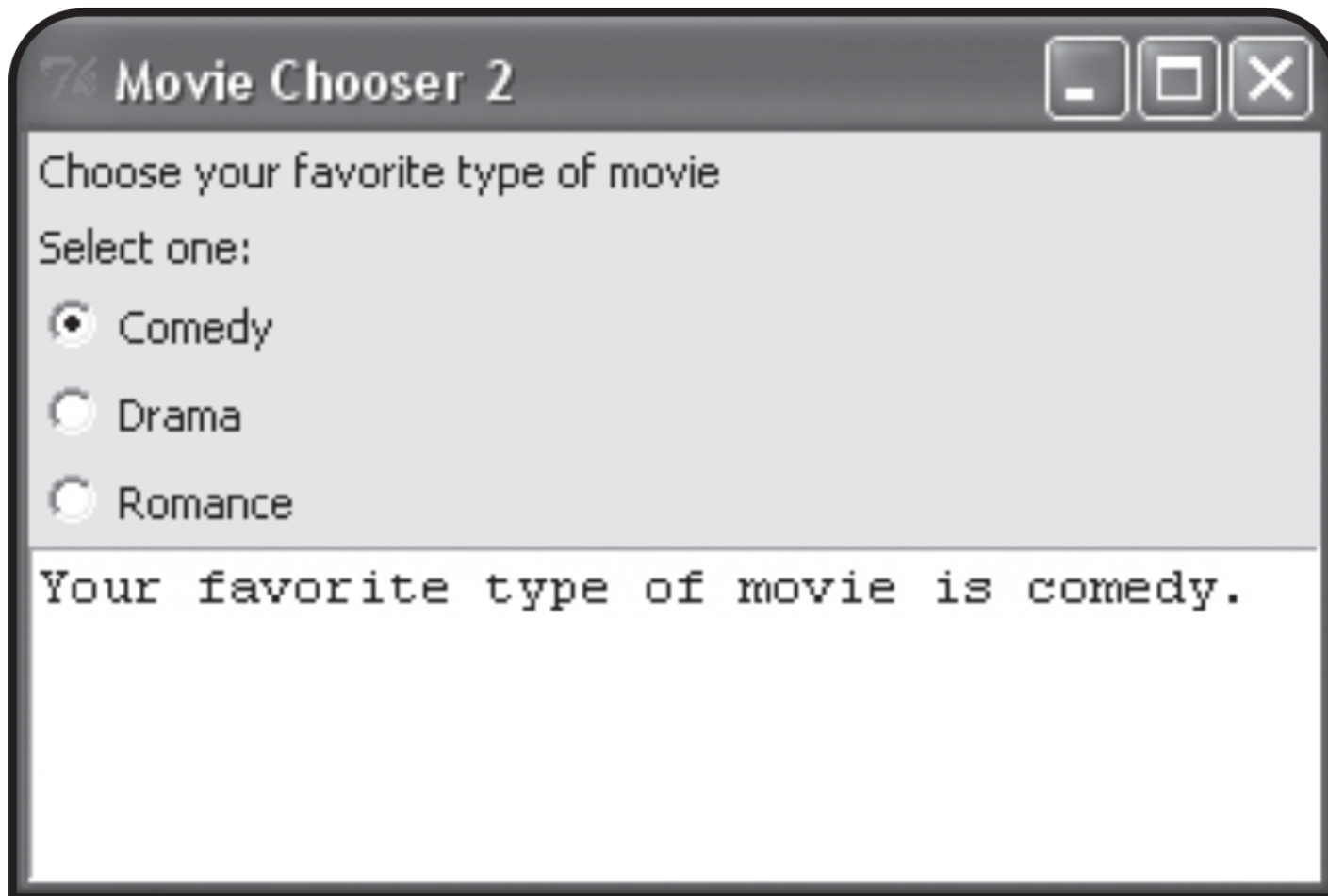
- You can't access the value of a `BooleanVar` object directly. Instead, you must invoke the object's `get()` method.

- We use `get()` of the `BooleanVar` object referenced by `likes_comedy` to get the object's value.
- If the value evaluates to `True`, so the Comedy check button is selected, and we add `"You like comedic movies.\n"` to the string we are building to display in the text box.
- Perform similar operations based on the status of the Drama and Romance check buttons.
- Delete all of the text in the text box and then insert the new string, `likes`.

# Introducing the Movie Chooser 2 Program

The batch file: [movie\\_chooser2.bat](#)

[movie\\_chooser2.py](#)  
**pause**



# movie\_chooser2.py

```
# Movie Chooser 2  
# Demonstrates radio buttons
```

```
from tkinter import *
```

```
class Application(Frame):
```

```
    """ GUI Application for favorite movie type. """
```

```
    def __init__(self, master):
```

```
        """ Initialize Frame. """
```

```
        super(Application, self).__init__(master)
```

```
        self.grid()
```

```
        self.create_widgets()
```

```
    def create_widgets(self):
```

```
        """ Create widgets for movie type choices. """
```

```
        # create description label
```

```
        Label(self, text = "Choose your favorite type of \\  
movie").grid(row = 0, column = 0, sticky = W)
```

```
# create instruction label
Label(self, text = "Select one:"\
      ).grid(row = 1, column = 0, sticky = W)
```

```
# variable for single, favorite type of movie
self.favorite = StringVar()\
self.favorite.set(None)
```

```
# create Comedy radio button
Radiobutton(self, text = "Comedy",\
            variable = self.favorite, value = "comedy.",\
            command = self.update_text\
            ).grid(row = 2, column = 0, sticky = W)
```

```
# create Drama radio button
Radiobutton(self, text = "Drama",\
            variable = self.favorite, value = "drama.",\
            command = self.update_text\
            ).grid(row = 3, column = 0, sticky = W)
```



```
# create Romance radio button
Radiobutton(self, text = "Romance",
            variable=self.favorite, value="romance.",
            command = self.update_text\
            ).grid(row = 4, column = 0, sticky = W)
```

```
# create text field to display result
self.results_txt = Text(self, width=40, height=5,
                        wrap = WORD)
self.results_txt.grid(row=5, column=0,
                      colspan=3)
```

```
def update_text(self):
    """ Update & display user's favorite movie type """
    message = "Your favorite type of movie is "
    message += self.favorite.get()

    self.results_txt.delete(0.0, END)
    self.results_txt.insert(0.0, message)
```

```
# main  
root = Tk()  
root.title("Movie Chooser 2")  
app = Application(root)  
root.mainloop()
```

# Creating Radio Buttons

- Since only one radio button in a group can be selected at one time, there's no need for each radio button to have its own status variable, as required for check buttons.
- A group of radio buttons share one, special object that reflects which of the radio buttons is selected. This object can be an instance of the **StringVar** class from the **tkinter** module, which allows a string to be stored and retrieved.
- Before we create the radio buttons, we create a single **StringVar** object for all of the radio buttons to share, assign it to the attribute **favorite**, and set its initial value to **None** using the object's **set()** method:

```
self.favorite = StringVar()  
self.favorite.set(None)
```

- Create the Comedy radio button:

```
Radiobutton(self, text = "Comedy",  
             variable = self.favorite, value = "comedy.",  
             command = self.update_text\  
             ).grid(row = 2, column = 0, sticky = W)
```

- A radio button's `variable` option defines the special variable associated with the radio button, while a radio button's `value` option defines the value to be stored by the special variable when the radio button is selected.
- By setting this radio button's `variable` to `self.favorite` and its `value` to `"comedy."`, we're saying that when the Comedy radio button is selected, the **StringVar** referenced by `self.favorite` should store the string `"comedy."`
- Create the other 2 radio buttons:

```
Radiobutton(self, text = "Drama",  
            variable = self.favorite, value = "drama.",  
            command = self.update_text\  
            ).grid(row = 3, column = 0, sticky = W)
```

```
Radiobutton(self, text = "Romance",  
            variable = self.favorite, value="romance.",  
            command = self.update_text\  
            ).grid(row = 4, column = 0, sticky = W)
```

- By setting the Drama/Romance radio button's `variable` to `self.favorite` and its `value` to `"drama."/``"romance."`, when the Drama/Romance radio button is selected, the `StringVar` referenced by `self.favorite` should store `"drama."/``"romance."`
- Create the text box to display the results:

```
self.results_txt = Text(self, width = 40, height = 5,  
                        wrap = WORD)  
self.results_txt.grid(row=5,column=0,columnspan=3)
```

# Getting a Value from a Group of Radio Buttons

- Getting a value from a group of radio buttons is to invoke the `get()` method of the `StringVar` object that they all share:

```
def update_text(self):  
    message = "Your favorite type of movie is "  
    message += self.favorite.get()
```

- When the Comedy/Drama/Romance radio button is selected, `self.favorite.get()` returns "comedy. "/"drama. "/"romance."
- Delete any text in the text box and insert the string which declares the user's favorite movie type:

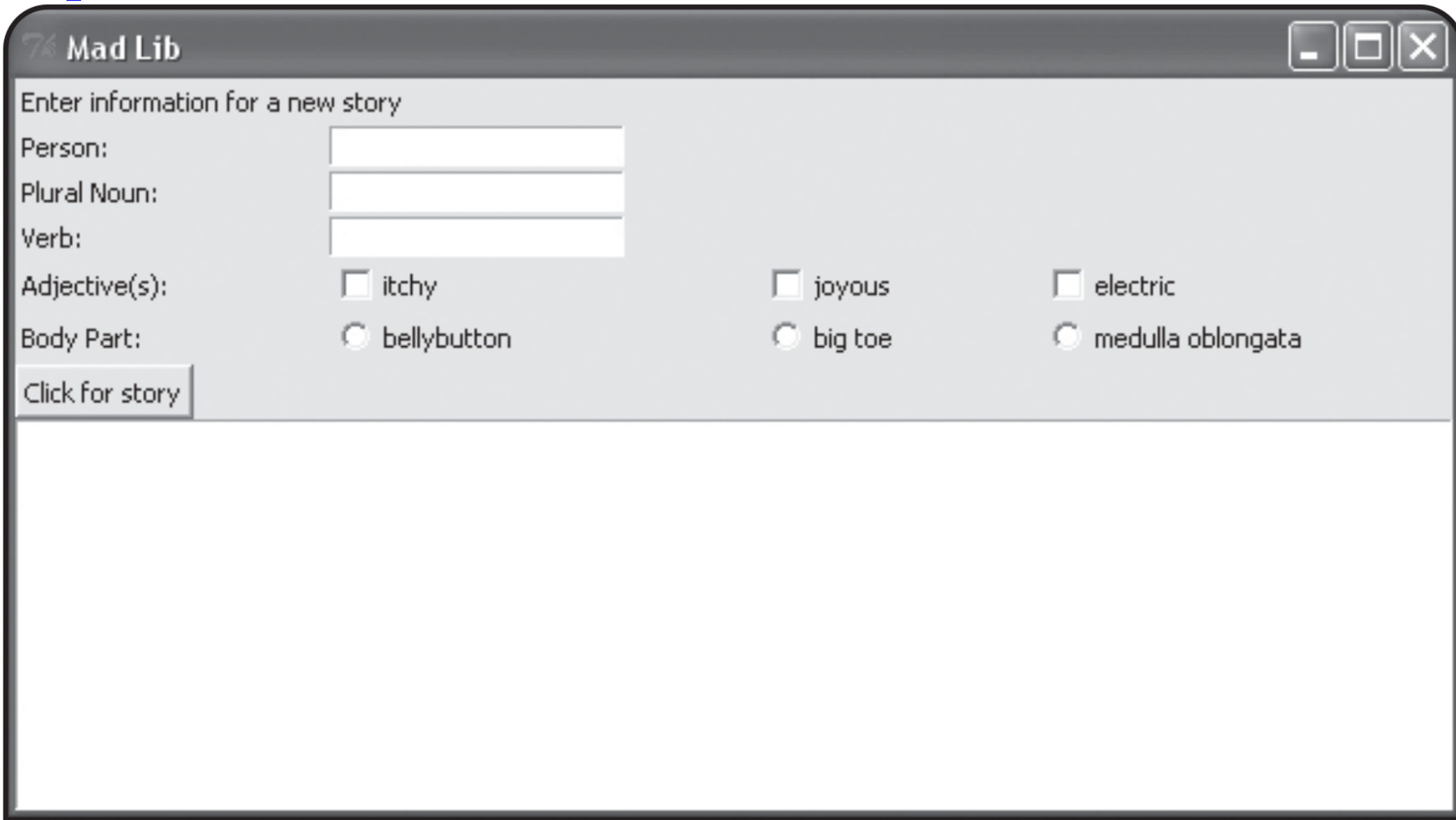
```
self.results_txt.delete(0.0, END)  
self.results_txt.insert(0.0, message)
```

# Introducing the Mad Lib Program

The batch file: [mad\\_lib.bat](#)

[mad\\_lib.py](#)

[pause](#)



The screenshot shows a window titled "Mad Lib" with a standard Windows-style title bar (minimize, maximize, close buttons). The window content is a form for generating a mad lib story. It includes labels for "Person:", "Plural Noun:", "Verb:", "Adjective(s):", and "Body Part:". Each label is followed by a text input field or a set of radio buttons. The "Adjective(s)" and "Body Part:" labels have three options each, each with a radio button. A "Click for story" button is located at the bottom left of the form area. The rest of the window is a large empty space, presumably for the generated story.

Mad Lib

Enter information for a new story

Person:

Plural Noun:

Verb:

Adjective(s):  itchy  joyous  electric

Body Part:  bellybutton  big toe  medulla oblongata

Click for story

# 76 Mad Lib



Enter information for a new story

Person:

Plural Noun:

Verb:

Adjective(s):  itchy

joyous

electric

Body Part:  bellybutton

big toe

medulla oblongata

[Click for story](#)



## 76 Mad Lib



Enter information for a new story

Person:

Plural Noun:

Verb:

Adjective(s):  itchy  joyous  electric

Body Part:  bellybutton  big toe  medulla oblongata

[Click for story](#)

The famous explorer Keren had nearly given up a life-long quest to find The Lost City of Salamanders when one day, the salamanders found Keren. A strong, itchy, electric, peculiar feeling overwhelmed the explorer. After all this time, the quest was finally over. A tear came to Keren's big toe. And then, the salamanders promptly devoured Keren. The moral of the story? Be careful what you dance for.

# mad\_lib.py

```
# Mad Lib
```

```
# Create a story based on user input
```

```
from tkinter import *
```

```
class Application(Frame):
```

```
    """ GUI app creates a story based on user input. """
```

```
    def __init__(self, master):
```

```
        """ Initialize Frame. """
```

```
        super(Application, self).__init__(master)
```

```
        self.grid()
```

```
        self.create_widgets()
```

```
    def create_widgets(self):
```

```
        """ Create widgets for story info & display. """
```

```
        # create instruction label
```

```
        Label(self, text="Enter information for a new story"\  
                ).grid(row=0, column=0, columnspan=2, sticky=W)
```

```
# create a label & text entry for the person's name.
```

```
Label(self, text = "Person: "\n\n        ).grid(row = 1, column = 0, sticky = W)\nself.person_ent = Entry(self)\nself.person_ent.grid(row = 1, column=1, sticky = W)
```

```
# create a label and text entry for a plural noun
```

```
Label(self, text = "Plural Noun:"\n\n        ).grid(row = 2, column = 0, sticky = W)\nself.noun_ent = Entry(self)\nself.noun_ent.grid(row = 2, column = 1, sticky = W)
```

```
# create a label and text entry for a verb
```

```
Label(self, text = "Verb:"\n\n        ).grid(row = 3, column = 0, sticky = W)\nself.verb_ent = Entry(self)\nself.verb_ent.grid(row = 3, column = 1, sticky = W)
```

```
# create a label for adjectives check buttons
```

```
Label(self, text = "Adjective(s):"\n\n        ).grid(row = 4, column = 0, sticky = W)
```

```
# create itchy check button
self.is_itchy = BooleanVar()
Checkbutton(self, text="itchy", variable=self.is_itchy\
        ).grid(row = 4, column = 1, sticky = W)
```

```
# create joyous check button
self.is_joyous = BooleanVar()
Checkbutton(self, text = "joyous",
        variable = self.is_joyous\
        ).grid(row = 4, column = 2, sticky = W)
```

```
# create electric check button
self.is_electric = BooleanVar()
Checkbutton(self, text = "electric",
        variable = self.is_electric\
        ).grid(row = 4, column = 3, sticky = W)
```

```
# create a label for body parts radio buttons
Label(self, text = "Body Part:"\
        ).grid(row = 5, column = 0, sticky = W)
```

```
# create variable for single, body part
```

```
self.body_part = StringVar()
```

```
self.body_part.set(None)
```

```
# create body part radio buttons
```

```
body_parts = ["bellybutton", "big toe",  
              "medulla oblongata"]
```

```
column = 1
```

```
for part in body_parts:
```

```
    Radiobutton(self, text=part,
```

```
    variable=self.body_part, value=part).grid(row=5,
```

```
    column=column, sticky=W)
```

```
    column += 1
```

```
# create a submit button
```

```
Button(self, text = "Click for story",
```

```
    command = self.tell_story\
```

```
    ).grid(row = 6, column = 0, sticky = W)
```

```
self.story_txt=Text(self,width=75,height=10,
```

```
    wrap=WORD)
```

```
self.story_txt.grid(row=7,column=0, columnspan=4)
```

```
def tell_story(self):  
    """Fill text box with story based on user input. """  
    # get values from the GUI  
    person = self.person_ent.get()  
    noun = self.noun_ent.get()  
    verb = self.verb_ent.get()  
    adjectives = ""  
    if self.is_itchy.get():  
        adjectives += "itchy, "  
    if self.is_joyous.get():  
        adjectives += "joyous, "  
    if self.is_electric.get():  
        adjectives += "electric, "  
    body_part = self.body_part.get()  
  
    # create the story  
    story = "The famous explorer "  
    story += person  
    story += " had nearly given up a life-long quest" + \  
        " to find The Lost City of "  
    story += noun.title()
```

**story += " when one day, the "**  
**story += noun**  
**story += " found "**  
**story += person + ". "**  
**story += "A strong, "**  
**story += adjectives**  
**story += "weird feeling overwhelmed the explorer. "**  
**story += "After all this time, the quest was " + \**  
**"finally over. A tear came to "**  
**story += person + "'s "**  
**story += body\_part + ". "**  
**story += "And then, the "**  
**story += noun**  
**story += " promptly devoured "**  
**story += person + ". The "**  
**story += "moral of the story? Be careful what you"**  
**story += verb**  
**story += " for."**

```
# display the story  
self.story_txt.delete(0.0, END)  
self.story_txt.insert(0.0, story)
```

```
# main  
root = Tk()  
root.title("Mad Lib")  
app = Application(root)  
root.mainloop()
```



## **The Application Class's create\_widgets()**

- This class creates all of the widgets in the GUI. The only new thing is to create all 3 radio buttons in a loop by moving through a list of strings for each radio button's text and value options. (See codes.)

## **The Application Class's tell\_story()**

- In this method, we get the values the user has entered and use them to create the one, long string for the story. Then, we delete any text in the text box and insert the new string to show the user the story he or she created. (See codes.)