Lecture Notes

Overview

Chapter 9 provides an introduction to graphical user interfaces. Students learn about the structure of a GUI-based program and using the MVC pattern. Through several examples, students learn to instantiate and lay out different types of window objects, including labels, entry fields, and command buttons, in a window's frame. Events and event-driven programming are also introduced, and students learn to define methods that handle events associated with window objects. Achieving different layouts using grids and nested frames (panes) is also covered.

Objectives

After completing this chapter, students will be able to:

- Structure a GUI-based program using the model/view/controller pattern
- Instantiate and lay out different types of window objects such as labels, entry fields, and command buttons
- Define methods that handle events associated with window objects
- Organize sets of window objects in nested frames

The Behavior of Terminal-Based Programs and GUI-Based Programs

This section presents two different versions of the bouncy program from a user's point of view: a terminal-based user interface and a *graphical user interface*.

The Terminal-Based Version

Use Figure 9.1 for a description of how the user interacts with the terminal-based version of the program.

There are different effects of this interface on users, as listed in the bullet points on Page 351 of the text.

The GUI-Based Version

Use Figure 9.2 for a description of how the user interacts with the GUI-based version of the program. Introduce the following terms: *window object*, *widget*, *label*, *entry field*, *command button*, and *title bar*.

This interface can be used to solve the user interaction problems introduced with the terminal-based version.

Event-Driven Programming

Review the role of *event-driven programming* in GUI-based programs, and identify the steps involved in such programming.

GUI based programs adhere to the model/view pattern, which separates the resources and responsibilities for managing the data model from those concerned with displaying it and interacting with users.

Use Figure 9.3 to review the *model/view/controller* (*MVC*) pattern. The PyjamasDesktop Python toolkit uses the PyWebkitGtk toolkit and provides the "V" in the MVC pattern. For more information, visit: http://wiki.python.org/moin/PyjamasDesktop.

Review how to code a GUI-based event-driven program (see the steps listed on pages 354-355).

Coding Simple GUI-Based Programs

Review the role of the tkinter and tkinter.messagebox standard Python modules. Python has a huge number of GUI frameworks (or toolkits) available for it. You can find a list of several of these toolkits at: http://wiki.python.org/moin/GuiProgramming.

tkinter is based on the Tk widget set (a package to implement GUI programs under a windowing system). For more information on tkinter, visit: http://infohost.nmt.edu/tcc/help/lang/python/tkinter.html.

Windows and Labels

Use the LabelDemo example provided in the book and Figure 9.4 for an description of how to create windows and labels using tkinter. Review the term *grid layout* and remind students of the meaning of the term *parent component*.

Displaying Images

Use the ImageDemo example provided in the book and Figure 9.5 to see how to create a label with an image using tkinter.

Command Buttons and Responding to Events

Use the ButtonDemo example provided in the book and Figure 9.6 to see how to create a button and how to enable it to respond to clicks.

Viewing the Images of Playing Cards

Use Figure 9.7 and the code provided in the book to see how to view the images of playing cards in Python. The Card and Deck classes created in Chapter 8 are reused for this purpose, and the program uses existing image files to display the cards.

Entry Fields for the Input and Output of Text

Review the terms: *form filler* and *entry field*.

Use Table 9.1 for a description of the three types of data container objects that can be used with Entry fields.

Use the CircleArea example and Figure 9.8 to see how to use Entry fields.

Using Pop-up Dialog Boxes

Use Table 9.2 as an introduction to the role of some of the most important tkinter.messagebox functions.

Use Figure 9.9 and the sample code provided in the book to see how to create an event handler that uses a pop-up dialog box that shows an error message. For more information on the tkinter.messagebox module, visit: http://epydoc.sourceforge.net/stdlib/tkMessageBox-module.html.

Colors

The RGB color system can be used with tkinter (both using the hex notation and the predefined string values).

You can set two attributes for most GUI components: a foreground color (fg) and a background color (bg).

Text Attributes

Use Table 9.3 and one or more examples for an explanation of how to modify a *type font*.

Sizing and Justifying an Entry

Use the sample code provided in the book and Figure 9.12 to see how to set the size and justification of entry fields.

Sizing the Main Window

Review the role of the geometry and resizable methods that can be run with the root window to affect its sizing.

Grid Attributes

Note that, by default, a newly opened window shrink-wraps around its components and is resizable.

Use Table 9.4 and the examples provided in the book, for a description of the Grid attributes and how to use them.

Review the term *expansion weight* and how an expansion weight can be assigned to a row or column of cells. For more information on the tkinter grid geometry manager, visit: http://effbot.org/tkinterbook/grid.htm.

"TK provides three geometry managers: .pack(), .grid() and .place()." For more information, visit: www.ibm.com/developerworks/linux/library/l-tkprg/.

Using Nested Frames to Organize Components

Use the ComplexLayout example and Figure 9.16 to see how to use nested frames (*panes*) to organize GUI components.

Multi-Line Text Widgets

Use the TextDemo example and Figure 9.17 to see how to create multi-line Text widgets.

Scrolling List Boxes

Review the term *list box*, and use Table 9.5 for a description of some of the most useful Listbox methods.

Using Figure 9.18 and the sample code provided in the book, review how to use these methods to create scrolling list boxes.

Mouse Events

Use Table 9.6 for an introduction to the mouse events available in Python.

Review how to associate a mouse event and an event-handling method with a widget by calling the bind method.

Keyboard Events

Use Table 9.7 for an introduction to some of the most useful key events available in Python.

Research how to bind a keyboard event to a handler. For more information on events and bindings in Python, read: www.pythonware.com/library/tkinter/introduction/events-and-bindings.htm.

Additional Resources

1. GUI Programming in Python:
   http://wiki.python.org/moin/GuiProgramming

2. Tkinter: GUI programming with Python:
   http://infohost.nmt.edu/tcc/help/lang/python/tkinter.html

3. Graphic User Interface FAQ:
   www.python.org/doc/faq/gui/

4. Module tkMessageBox:
   http://epydoc.sourceforge.net/stdlib/tkMessageBox-module.html

5. The Tkinter Grid Geometry Manager:
   http://effbot.org/tkinterbook/grid.htm

6. Events and Bindings:
   www.pythonware.com/library/tkinter/introduction/events-and-bindings.htm

Key Terms

- **button object:** A window object that allows the user to select an action by clicking a mouse.
- **event:** An occurrence, such as a button click or a mouse motion, that can be detected and processed by a program.
- **event-driven loop:** A process, usually hidden in the operating system, that waits for an event, notifies a program that an event has occurred, and returns to wait for more events.
- **grid layout:** A Python layout class that allows the user to place window objects in a two-dimensional grid in the window.
- **GUI (graphical user interface):** A means of communication between human beings and computers that uses a pointing device for input and a bitmapped screen for output. The bitmap displays images of windows and window objects such as buttons, text fields, and drop-down menus. The user interacts with the interface by using the mouse to directly manipulate the window objects. See also window object.
- **justification:** The process of aligning text to the left, the center, or the right within a given number of columns.
- **label object:** A window object that displays text, usually to describe the roles of other window objects.
- **model/view/controller pattern (MVC):** A design plan in which the roles and responsibilities of the system are cleanly divided among data management (model), user interface display (view), and user event-handling (controller) tasks.
- **window object (widget):** A computational object that displays an image, such as a button or a text field, in a window and supports interaction with the user.