PC204 Lecture 9

Conrad Huang
conrad@cgl.ucsf.edu
Genentech Hall, N453A
x6-0415

Topics

- Homework review
- Interactive Programs *vs* Scripts
- Tkinter
- Pmw

Homework Review

- 8.1 – poker probabilities
- 8.2 – short project description

Interactive Program *vs* Script

- "Script" is a program where the programmer determines the order of execution from beginning to end
  - Programs may need user input and still be scripts, eg they may ask the user to type some input at a particular point of execution
- "Interactive program" is a program where the user determines the order of execution from user input using either:
  - a command-based text interface
  - a graphical user interface (GUI, pronounced goo-ey)

Interactive Program Characteristic

- The mark of an interactive program is an event dispatch loop
- For a text-based interactive program, it looks something like this

```
for line in sys.stdin:
        args = line.strip().split()
        if not args:
                # skip blank lines
                continue
        if args[0] == “cmd1”:
                cmd1(args[1:])
        elif args[0] == “cmd2”:
                cmd2(args[1:])
        else:
                print “unknown command:”, args[0]
```

Event Loop and Event Dispatch

- The **for** loop reading user input is called an **event loop** because it waits for user events
- The event loop **dispatches** the user input to functions based on the input
- The event loop and dispatcher may be abstracted into a simple textual user interface module (tui.py)

```
_registry = {}

def mainloop(prompt="> "):
    import sys
    sys.stdout.write(prompt)
    while True:
        line = sys.stdin.readline()
        if not line:
            break
        args = line.strip().split()
        if not args:
            # ignore empty lines
            continue
        ambiguous = False
        f = None
        for k in _registry.keys():
            if k.startswith(args[0]):
                if f is not None:
                    ambiguous = True
                    break
                f = _registry[k]
        if ambiguous:
            print ("Error: %s is an ambiguous command" % args[0])
        elif f is None:
            print ("Error: %s is an unkown command" % args[0])
        else:
            if not f(args[1:]):
                break
        sys.stdout.write(prompt)

def register(function):
    _registry[function.__name__] = function
```

Using the TUI Module

```
def test(args):
    print "test", args
    return True

def tilt(args):
    print "tilt", args
    return True

def verify(args):
    print "verify", args
    return True

def quit(args):
    print "quit"
    return False

import tui
tui.register(test)
tui.register(tilt)
tui.register(verify)
tui.register(quit)
tui.mainloop()
print "Returned from command loop"
```

- usetui.py is an example of how to use the TUI module
  - Define functions that handle events
  - Register functions with TUI module
  - Enter the TUI module event loop
- tui module may be reused for many scripts

Libraries and Frameworks

- Library functions are called by user code to perform some tasks and return after the tasks are completed
- **tui.mainloop()** is somewhat different from most library functions in that it **calls back** to user-registered functions before returning
- A library that defines most functionality but allows callers to plug in their own code that gets called at predefined times is often called a **framework**

GUI Frameworks

- Most programs using a GUI framework follows the same form as our TUI program
  - Define callback functions
  - Define user interface elements, aka **widgets**, and register callback functions
  - Enter GUI event loop
- Implementation-wise, you need to design your user interface before you know what callback functions you need to make the program work

Tkinter

- Tkinter is a GUI framework
- Common widgets in Tkinter include:
  - Frame – placeholder for placement of widgets
  - Label – widget for displaying some text
  - Button – push button that user may press
  - Entry – input widget where user may type
  - Canvas – graphical drawing area
- A callback may be registered for a widget
  - eg Button callback is invoked when the button is pushed
- A function may also be bound to a widget to respond to events
  - eg Function gets called when user presses return in an entry widget

Basic Tkinter Widgets

- Using the Button widget
  - tkinter_button.py
  - tkinter_button2.py
  - tkinter_button3.py
- Using the Label and Entry widgets
  - tkinter_label.py
  - tkinter_label_entry.py
- Controlling layout using Frame widgets
  - tkinter_label_entry2.py
- Binding actions to events
  - tkinter_label_entry3.py

Canvas and Text

- Displaying graphics on a canvas
  - tkinter_canvas.py
- A design pattern for a GUI class
  - tkinter_canvas2.py
- Useful for homework assignment
  - tkinter_canvas3.py
- Displaying and editing text
  - tkinter_text.py

More Tkinter Widgets

- Tkinter offers many more widgets
  - Some are simple, eg checkbutton and scale
  - Some are complex, eg scrollbars and menu buttons
- For more complete documentation on Tkinter widgets, see *An Introduction to Tkinter* by Fredrik Lundh
  - http://www.pythonware.com/library/tkinter/introduction/

More Complex Widgets

- Tkinter is a low-level interface that offers a lot of programming flexibility
- Libraries built on top of Tkinter offer more functionality with less programming
  - For example, managing scrollbars for canvas or text widgets is possible with Tkinter, but painful
- Python MegaWidgets (PMW) is layered on top of Tkinter and implements high(er)-level widgets such as ScrolledCanvas and NoteBook

Pmw

- Pmw (http://pmw.sourceforge.net/) is a Python package containing a number of "megawidgets" that are composed of basic Tkinter widgets
- To use PMW, you need to download and install the Python code
  - Version 1.3 of PMW may be fetched locally
    - http://www.cgl.ucsf.edu/Outreach/pc204/lecture_notes/week9/Pmw-1.3.3b.zip
  - The simplest way to use this is to extract the "Pmw" folder and put it with your source code

An Aside on Packages

- A Python package is a folder of related Python modules
  - A Python package folder must contain a file named **__init__.py**, which
    - tells Python the folder is not a random collection of .py files
    - is executed when the package is imported
  - All other .py files in the folder are considered modules of the package and may be imported with
    - **import *package_name.module_name***
- Pmw uses some serious Python magic for its package management, so I usually just follow the recipes in the excellent documentation

Pmw (cont.)

- pmw_scrolledcanvas.py
  - Uses canvas with attached scrollbars
  - Uses menu buttons for scrollbar modes
  - Uses button box for array of aligned buttons

Wrapping Up

- There are better ways for constructing GUIs
  - WYSIWYG interface design tools
  - More comprehensive widget sets
- Tkinter and Pmw together provide a usable set of tools for building graphical user interfaces
- A lot of applications are migrating to a client-server model that uses a web browser as a front end
  - GUI (client end) is written in HTML5 and Javascript
  - Server end is written in a variety of languages, including Python, PHP, Perl, …

Debugging

- Don't guess
  - Collect data on where the error is occurring
    - read the traceback
    - add invariant, pre- and post-condition checks
    - use print statements
  - Change one thing at a time
- Don't get discouraged
  - No, the computer is ***not*** out to get you
  - Explain the code to someone (perhaps yourself)
  - Take a break
    - don't work until you can't keep your eyes open

Homework and Project

- Assignment 9.1
  - tkinter_canvas3.py should be helpful
- Assignment 9.2
  - A fuller description of your final project