The current topic: Tkinter

- Introduction
- Object-oriented programming: Python
- Functional programming: Scheme
  - Introduction
  - Numeric operators, REPL, quotes, functions, conditionals
  - Function examples, helper functions, let, let*
  - More function examples, higher-order functions
  - More higher-order functions, trees
  - More trees, lambda reductions, mutual recursion, examples, letrec
- Python GUI programming (Tkinter)
- Types and values
- Logic programming: Prolog
- Syntax and semantics
- Exceptions

Announcements

- Lab 2 was due today at 10:30 am.
- Reminder: Term Test 2 is on Monday November 3rd in GB405, not in the regular lecture room.
  - 50 minutes (11:10 – 12:00).
  - You’re allowed to have one double-sided aid sheet for the test. You must use standard letter-sized (that is, 8.5” x 11”) paper. The aid sheet can be produced however you like (typed or handwritten).
  - Bring your TCard.
  - What’s covered?
    - Everything from September 29 up to and including October 24.
    - Lab 2.
  - An old Term Test 2 has been posted.
  - The exercises at the end of each lecture are also good practice.

What is Tkinter?

- Tkinter is a Python interface to Tk.
  - Tkinter = Tk interface

- Tk is a cross-platform GUI library.
  - It lets you write GUI code that runs (without modification) on Windows, Linux, Mac OS, etc.
  - Tk uses native widgets on each platform, so your program gets a native look-and-feel on each platform even though you’ve only written the code once.
    - That is, it looks like a Windows application when run on Windows, it looks like a Mac application when run on Mac OS, etc., without any extra programming effort.

Overview

- This lecture is meant to be just an introduction to Tkinter.
  - It will not include everything you need for the project.
  - Learning Tkinter is still part of the project.

- We’ll look at:
  - The basic structure of Tkinter code.
  - Some of the Tkinter widgets that you might find useful.
  - Registering callback functions.
A simple Tkinter program

```python
import Tkinter
root = Tkinter.Tk()
Tkinter.mainloop()
```

- Observe that:
  - You need to import the `Tkinter` module.
  - Calling `Tkinter.Tk()` creates a "root" (top-level) window.
  - Calling `mainloop()` starts Tkinter's event loop. Without this call, nothing will be displayed (unless you're working in the Python interpreter).
  - The call to `mainloop()` does not return until all Tkinter windows are closed. This means that any code after this call won't get executed until all windows are closed.

Another simple Tkinter program

```python
import Tkinter
root = Tkinter.Tk()
root.title("Hello!")
myLabel = Tkinter.Label(root, text="CSC326", fg="blue", bg="yellow")
myLabel.pack()
Tkinter.mainloop()
```

# The following produces an error since it isn't run till the window is closed.
label2 = Tkinter.Label(root, text="Project", fg="green")
label2.pack()

The Tkinter event loop

- Calling `mainloop()` starts Tkinter's event loop.

  - In the event loop, Tkinter "listens" for and responds to particular events (like clicks and key presses).
    - This is called *event-driven programming*. Instead of a sequence of steps that is fixed ahead of time by the programmer, the program's behaviour depends on events that occur at runtime.
    - We'll see later that we can register callback functions that act in response to particular events.

  - In the event loop, Tkinter periodically redraws windows.
    - No drawing occurs outside of the event loop, so nothing appears on the screen before the call to `mainloop()`.

  - Since the call to `mainloop()` does not return until all windows are closed (which, essentially, is the end of your program), you need to do all initial setup of your GUI *before* the call to `mainloop()`.

Creating widgets

- When creating a widget (for example, a Label), we need to specify its parent (where it's going to go).
  - We can optionally specify other properties using keyword arguments.
    ```python
    myLabel = Tkinter.Label(root, text="CSC326", fg="blue", bg="yellow")
    ```
  - The above line creates a Label widget whose parent is the root window.
  - It also specifies the Label's text, foreground colour, and background colour.
  - It does not actually place the Label on the window. The Label is placed on the window by the line:
    ```python
    myLabel.pack()
    ```

  - Notice that we're not specifying where on the window the Label should go. The default is to place the first widget on top, the next one underneath it, and so on, from top to bottom.
import Tkinter

root = Tkinter.Tk()
root.title("Hello!")

myLabel = Tkinter.Label(root, text="CSC326", fg="blue")
myLabel.pack(side=Tkinter.RIGHT)

label2 = Tkinter.Label(root, text="Project", fg="red")
label2.pack(side=Tkinter.LEFT)

Tkinter.mainloop()
import Tkinter
root = Tkinter.Tk()
root.title("Buttons!")
myButton = Tkinter.Button(root, text="CSC326")
myButton.pack()
root.mainloop()

• The button can be clicked, but it doesn’t do anything (yet).

Responding to a button click

import Tkinter
root = Tkinter.Tk()
root.title("Buttons!")
def changeCol():
    root.config(bg="blue")
myButton = Tkinter.Button(root, text="CSC326", command=changeCol)
myButton.pack()
root.mainloop()

• Clicking the button changes the window’s colour to blue.
• Observe that we used the keyword argument command to set the function changeCol as a callback function that is called whenever the button is clicked.

Responding to button clicks

import Tkinter
def blueCol():
    root.config(bg="blue")
def redCol():
    root.config(bg="red")
root = Tkinter.Tk()
root.title("Buttons!")
blueBtn = Tkinter.Button(root, text="Blue!", command=blueCol)
redBtn = Tkinter.Button(root, text="Red!", command=redCol)
blueBtn.pack()
redBtn.pack()
root.mainloop()

• Problem: Code duplication (redCol and blueCol).
Callbacks and arguments

- Observe that the code duplication problem gets worse if we add buttons for more colours (green, orange, etc.).

- To solve this problem, we need to have just a single function, changeCol, that takes a colour as an argument and sets the background to this colour.
  - But the callback function for a button is called with no arguments.
  - So we can’t set such a changeCol function as the callback for a button.

- Solution: lambda expressions!
  - Just like in Scheme, lambda expressions in Python create functions.
  - However, in Python, the body of a lambda expression must be just a single expression, so lambda expressions cannot always replace a def statement.
  - Python lambda expression syntax:
    

        (lambda arg1, arg2, ...: expression)

  For example:

        (lambda x, y: x+y) (3, 4)   # This evaluates to 7.

Using lambda to create a callback function

```
import Tkinter

def changeCol(colour):
    root.config(bg=colour)

root = Tkinter.Tk()
root.title("Buttons!")

blueB = Tkinter.Button(root, text="Blue!", command=(lambda: changeCol("blue")))
redB = Tkinter.Button(root, text="Red!", command=(lambda: changeCol("red")))
blueB.pack()
redB.pack()

root.mainloop()
```

- Observe that we’ve use lambda to create callback functions for each button.
  - These functions take no arguments (as required for button callbacks).
  - These functions call changeCol with an appropriate argument.

Canvasses

- The Canvas widget lets you draw various shapes.
  - Recall that you’re required to use a Canvas widget for the project’s game board.

- An example:

```
import Tkinter

root = Tkinter.Tk()
root.title("Canvas")

myC = Tkinter.Canvas(root)
myC.pack()

r = myC.create_rectangle(50,100,200,140, fill="blue")
l = myC.create_line(0,0,100,100, fill="red", width=2.0)

root.mainloop()
```

Canvas events

```
import Tkinter

root = Tkinter.Tk()
root.title("Canvas")

def makeRed(i):
    myC.itemconfig(i,fill="red")

myC = Tkinter.Canvas(root)
myC.pack()

r = myC.create_rectangle(50,100,200,140, fill="blue")
l = myC.create_line(0,0,100,100, fill="red", width=2.0)
myC.bind("<Button-1>", (lambda e: makeRed(r)))

root.mainloop()
```

- Clicking the canvas makes the rectangle red.
Canvas events

• Observe that the `bind()` method is used to set callbacks for a canvas.
  – The event called "<Button-1>" is a left-click.
  – A canvas callback function takes a single argument (we didn’t do anything with this argument).

• Observe that `itemconfig()` method can be used to configure objects that have already been added to the canvas.

• The argument passed to a canvas callback function is an event object that provides information about the event that caused the callback to be called.
  – For example, the location of the pointer on the canvas when the mouse was clicked.

Using canvas event objects

```python
import Tkinter
import tkMessageBox

root = Tkinter.Tk()
root.title("Canvas")

def announce(event):
    message = "Click at " + str((event.x, event.y))
    tkMessageBox.showinfo("Position", message)

myC = Tkinter.Canvas(root)
myC.pack()

r = myC.create_rectangle(50, 100, 200, 140, fill="blue")
l = myC.create_line(0, 0, 100, 100, fill="red", width=2.0)
myC.bind("<Button-1>", announce)
root.mainloop()
```

• Look at the next slide to see what happens when we click inside the canvas.

Using canvas event objects

• Observe that the x and y instance variables of the event object passed to a canvas callback function provide the canvas co-ordinates of the pointer when the event (in our case, a left-click) occurred.

• The `tkMessageBox` module provides a collection of dialog boxes (`showinfo`, `askyesno`, `askokcancel`, etc.).

• The call:
  ```python
tkMessageBox.showinfo("Position", message)
```
  creates a dialog box that displays the given `message` and has an OK button.
  – The first argument ("Position") is meant to set the title of the dialog box, but this seems to have no effect on Mac OS. Try this out on your system to see what happens.
Some other useful widgets

- Entry widgets
  - These allow the user to enter text.

- Radiobutton widgets
  - These allow the user to choose exactly one item from a collection.

- Listbox widgets
  - These allow the user to choose an item from a list.

- And more!

What we've left out

- Object-oriented design.
  - For the project, you need to create appropriate classes for your GUI.
  - And, more generally, you need to follow good programming style (unlike what we've seen in many of these examples).

- Tkinter.
  - There's (not surprisingly) much more than what we've covered in this lecture.
  - Widgets, methods, settings, etc.
  - This lecture was just meant to be a starting point.

Exercises

- Create a simple Tkinter application. Like we did in this lecture, start with an application that displays an empty window. Then add some widgets.

- Experiment with callback functions. First create a callback function for a button. Then create a callback function for a canvas.

- Go beyond what we did in this lecture. Read about some widgets that we didn’t cover (such as the widgets on slide 25), and add them to your application.