CSci 1133, Spring 2014
Lab Exercise 14

GUI Programming with Tk

Creating a Graphical User Interface (GUI) is often an important aspect of making easy-to-use, consumer oriented software. Common graphical input and display elements such as buttons, labels, and text boxes (collectively called widgets) can help a user quickly understand how to interact with your program. In this lab, we will use the Tk widget toolkit to create GUIs in Python.

Warm-up

In order to interact with the Tk widget toolkit, Python provides a module called `tkinter` that allows you to interface with Tk. In a new file, use the following code to import the module, and create a new window:

```
from tkinter import *

win = Tk()
```

Each call to `Tk()` creates a new **window**. We must save the window in the variable `win` so that we can edit its attributes. For example, we can edit attributes such as the size of the window as follows:

```
win.geometry("400x400")
```

This code uses the `geometry` method of a Tk window to set its size. The `geometry` method expects one argument, a string, in the form of `"WIDTHxHEIGHT"`, where `WIDTH` and `HEIGHT` are specified in units of pixels. Try making the window a different size.

There are several other attributes you may wish to change about your window. For now, let's just set the title of the window using the `title` command:

```
win.title("Super Fun Project")
```

Make sure to run your code anytime you edit it, to ensure everything works. You need to add `win.mainloop()` as the last line of code to make sure your program works.

Adding Widgets

So far, we have a window, but it's completely empty. Let's add some widgets to the window. Our first widget will be a label. **Labels** are used for displaying text, and can be created with the `Label` command:

```
greetings = Label(win,text="Hello World",font=("Helvetica",30))
greetings.place(x=50,y=100)
```

Labels have attributes just like windows. The first parameter in the constructor specifies which window the widget appears in, the rest allows us to set what text we would like the label to have, the typeface used and the size of the font. In the above code, we also use the `place` method of a label to set its position. Trying changing the font from Helvetica to Times, then choose a bigger font (remember to keep `mainloop()` as the last line). Can you see the difference between typefaces and font sizes?

One important note: If you don't place a widget after you create it, the widget will not be on the screen! You will notice in all of the following examples, every time we create a widget call `place` immediately after.

Our next widget will be the button. **Buttons** allow a user to click on them, and inform your code of these events using a callback function. A **callback** function is a function whose job it to respond when something has happened. Let's start by writing the callback function. In this case we want to change the text from "Hello, World" to "I've been clicked!". The following code shows how to change the text of the `greetings` label we made earlier.

```
def changeText():
    greetings["text"] = "I've been Clicked!"
```

Test the code by adding a call to `changeText()` as the last line of your program, you should see the new text in place of Hello Word. Now, remove this line of code as we don't want to see the text change until we click a button.

The code to add a button is very similar to adding a label:

```
testButton = Button(win, text="Click Me", command=changeText)
testButton.place(x=300,y=300)
```

This will create a button in the `win` window, with the text "Click Me", and register the function `changeText` as a callback. This means every time the button is clicked, `changeText` will be called. Before trying the button we need one more piece of code. Whenever you write GUI code using `tkinter` the last line of code in your program should always be a call to the special function `mainloop`.

```
mainloop() #always the last line of code
```

This function loops forever checking to see if any buttons have been clicked, keys pressed, mice moved, etc. Your code may sometimes work without this call (depending on OS), but it is safest to always include it as the last line. Go ahead and run your program and make sure the text changes when you click the button.

Next we will use the entry widget. The **Entry** widget allows you to create a text box where a user can input a single line of text. Add the widget with the following code:

```
nameEntry = Entry(win, width=25)
nameEntry.place(x=30,y=300)
```

This will place a text entry widget in our window `win`, with a width of 25 *characters* (not pixels). We also place the window in the bottom left to align with the button. Run the code, and you should be able to enter text into the text entry box.

We can access the contents of the box by calling the method. Let's *replace* the functionality of `changeText` to say hello to whoever's name is in the box:

```
def changeText():
    greetings["text"] = "Hello, "+nameEntry.get()
```

Run the code. After typing your name and pressing the button you should get a custom greeting. If your name is very long, you may have to move the `greetings` widget or choose a smaller font size.

You are now ready to construct more sophisticated GUI programs. The problems below can all be done using only Labels, Buttons, and Entry widgets. However, you may want to make more interesting variants using other widgets (e.g., check boxes, menus, radio buttons, slides, etc.).

Resources
To look into new widgets, or to get information on the ones we covered above you should check out this link:

http://effbot.org/tkinterbook/tkinter-classes.htm

You can read more about coloring and font options here:

http://effbot.org/tkinterbook/tkinter-widget-styling.htm (search for "Font descriptors")

Stretch

1). **User Specified Windows Size**
Write a program that uses the `input` command (twice) in order to ask the user for two integer values for width and height. Create a new window with the width and height as specified by the user. Hint: You will have to use some of the string processing commands we learned earlier such as: `str`, or `format`, or the `%` operator.

2). **Love Calculator**
Use two entry boxes to allow a user to type two names. Add a button that when pressed, places a score in a text box (a number from 1-100) based on how "compatible" the two names are. Make up any compatibility algorithm you like, but the displayed score must change when the names change.

Workout

1). **7-button calculator**

You can edit the text in an entry field using the `insert` and `delete` methods. Insert takes two parameters, the first is the position to insert at, and the second is the string the insert. Delete also takes two parameters, the beginning and end of where to delete. For both insert and delete you can use the special keyword END to specify the end of the text box. A simple example is given below:

```
from tkinter import *
win = Tk()
win.geometry("400x200")

def addX():
    eqn.insert(END,"X")

def clear():
    eqn.delete(0,END)

button1 = Button(win, text="Add X", command=addX)
button1.place(x=30,y=150)
buttonClear = Button(win, text="C", command=clear)
buttonClear.place(x=270,y=150)
eqn = Entry(win, width=25)
eqn.place(x=30,y=30)

mainloop()
```

For the Workout, create a simple calculator with 1 Entry box and the following 7 buttons:
1 – adds the number "1" to the Entry box
2 – adds the number "2" to the Entry box
3 – adds the number "3" to the Entry box
+ – adds the "+" symbol to the Entry box
- – adds the "-" symbol to the Entry box
C – clears the entry box
Eval – replaces the contents of Entry box with the value of the equation. This can be done easily using the built-in Python command: `eval(str)`.

2) Tic-tac-toe+

It's possible to change the text of a button using a button's `config()` method as follows:

```
myButton.config(text="Brand new button text")
```

Implement a tic-tac-toe game made with a 3-by-3 array of buttons. Each button should start off blank, but change it's text to "X" or "O" when it's clicked on. If there are 3 X's or O's in a row, that player wins. Notify the winner using a label.

If you have time, try this much-improved variant: create a 6 x 5 board, where a player must get 4 in a row to win. With this simple change, games between good players won't always end in a tie!

Challenge

If you've made it this far, you've made serious progress towards mastering a sophisticated widget toolkit, and you've build a good foundation for GUI programming. The best next step is to customize your existing widgets (can you change their colors?), or add new widgets (how might you add pictures?). Alternatively, you should try adding GUIs to some of our previous assignments like suggested below:

Encryption Machine

Create a program with: an Entry widget, a label, and two buttons, one labeled Encrypt, one Decrypt. After the user types in a word into the text box and presses the Encrypt button, apply a shift 3 substitution cipher to the text. If the user presses the Decrypt button, shift in the opposite direction and decipher the input text.

Full Calculator

Take your calculator in from the workout and create all the buttons 0-9, *, /, a decimal point button, and a sqrt button. Be sure to place all of the buttons in a logical order. Once you have that working, consider adding other functionality such as a memory button.