Visual Programming

College Of Computer Sciences and Mathematics

Dr. Firas Aswad

Dept. of Networks

Mar. 4, 2025

Binding Levels

We used an instance-level binding to bind an event to an instance of a widget, but if we want to bind an event to a particular class of widget, tkinter has levels of binding options:

- Application-level binding: It uses the same binding across all windows and widgets of an application as long as any window of the application is in focus.

```
widget.bind_all(event, callback, add=None)
```

```
root.bind_all('<F1>', show_help)
```

Irrespective of the widget currently under focus, pressing the F1 key will always trigger the show_help callback as long as the application is in focus.

Binding Levels

- Class-level binding: Use to bind events at a particular class level. This is normally used to set the same behavior for all instances of a particular widget class.

```
w.bind_class(class_name, event, callback, add=None)
```

```
my_entry.bind_class('Entry', '<Control-V>', paste)
```

all the entry widgets will be bound to the <Control-V> event, which will call a method named paste (event).

Lab work! bind a root and bind a frame.

Event Propagation

Most keyboard and mouse events occur at the operating system level.

The event propagates hierarchically upward from its source until it finds a window that has the corresponding binding.

The event propagation does not stop there. It propagates itself upwards, looking for other bindings from other widgets, until it reaches the root window.

If it does reach the root window and no bindings are discovered by it, the event is disregarded.

Handling Widget-Specific Variables

- In widgets, we need a string variable to track what the user enters into the entry widget or text widget.

- Also, Boolean variables to track whether the user has checked off the Checkbox widget.

- Integer variables to track the value entered in a Spinbox or Slider widget.

To respond to changes in widget-specific variables, Tkinter offers its own variable class.

The variable must be subclassed from this Tkinter variable class. Tkinter offers some commonly used predefined variables like StringVar, IntVar, BooleanVar, and DoubleVar.

These variables can be used to capture and play with the changes in the values of variables from within your callback functions. You can also define your own variable type if required.

Tkinter Variable

Creating a Tkinter variable is done by calling the constructor:

```
my_string = tk.StringVar()
ticked_yes = tk.BooleanVar()
group_choice = tk.IntVar()
volume = tk.DoubleVar()
```

Then, use it as a widget option, as follows:

```
tk.Entry(root, textvariable=my_string)
tk.Checkbutton(root, text="Remember Me", variable=ticked_yes)
tk.Radiobutton(root, text="Option1", variable=group_choice, value="option1")
tk.Scale(root, label="Volume Control", variable=volume, from =0, to=10)
```

Tkinter provides access to the values of variables via the set() and get() methods, as follows:

```
my_var.set("FooBar") # setting value of variable
my_var.get() # Assessing the value of the variable from say a callback
```

Lab work, run next code.

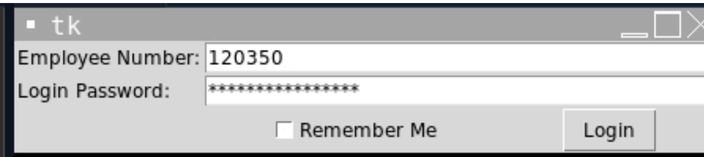
Demonstration: Tkinter Variable Class

main.py > ... Format

```python
"""
A demonstration of tkinter Variable Class
IntVar, StringVar & BooleanVar
"""
import tkinter as tk
root = tk.Tk()
def show():
  print("You entered:")
  print("Employee Number: " + str(employee_number.get()))
  print("Login Password: " + password.get())
  print("Remember Me: " + str(remember_me.get()))
  print('*' * 30)
#IntVar
tk.Label(root, text="Employee Number:").grid(row=1, column=1)
employee_number = tk.IntVar()
tk.Entry(root, width=40, textvariable=employee_number).grid(row=1,column=2,columnspan=2)
employee_number.set("120350")

#StringVar
tk.Label(root, text="Login Password:").grid(row=2, column=1, sticky='w')
password = tk.StringVar()  # defines the widget state as string
tk.Entry(root, width=40, show="*", textvariable=password).grid(row=2,column=2, columnspan=2)
password.set("mysecretpassword")

tk.Button(root, text="Login", command=show).grid(row=3, column=3)

#Boolean var
remember_me = tk.BooleanVar()
tk.Checkbutton(root, text="Remember Me", variable=remember_me).grid(row=3,column=2)
remember_me.set(True)

root.mainloop()
```

Run 18s on 15:36:57, 03/02

```
You entered:
Employee Number: 120350
Login Password: mysecretpassword
Remember Me: True
******************************
You entered:
Employee Number: 120350
Login Password: mysecretpassword
Remember Me: False
******************************
```



Discussion on events and callbacks

Summary

- Command binding
- Event binding
- The passing of extra arguments to a callback using the lambda function
- The binding of events to an entire application or a particular class of widget
- Using the Tkinter variable class to set and get the values of widget-specific variables

Unbinding and Virtual Events

Unbind: Tkinter provides the unbind option to undo the effect of an earlier binding.

```
widget.unbind(event) # Examples as:

entry.unbind('<Alt-Shift-5>')
root.unbind_all('<F1>')
root.unbind_class('Entry', '<KeyPress-Del>')
```

Virtual events: To create your own events. You can give these virtual events any name that you want. For example, let's suppose that you want to create a new event called <<commit>>, which is triggered by the F9 key. To create this virtual event on a given widget.

```
widget.event_add('<<commit>>', '<KeyRelease-F9>')
```

Unbinding and Virtual Events

You can then bind <<commit>> to a callback by using a normal bind() method, as follows:

```
widget.bind('<<commit>>', callback)
```

Style!

We can specify our own styling of widgets, such as their color, font size, border width, and relief. To specify widget options at the time of its instantiation, use this:

```
my_button = tk.Button(parent, **configuration options)
```

OR

```
my_button.configure(**options)
```

Specifying Styles

To specify the color options for a widget. You can specify the following two types of color for most widgets:

The background color
The foreground color

The commonly used representations are #rgb (4 bits), #rrggbb (8 bits), and #rrrgggbbb (12 bits).

For example, #fff is white, #000000 is black, #f00 is red (R=0xf, G=0x0 , B=0x0).


```
widget.configure (font='Times 8')
widget.configure(font='Helvetica 24 bold italic’)
```

Tkinter accepts five measurement units, px(pixel units), m(millimeters), c(centimeters), i(inches), and p(printer's points.

The default border width for most Tkinter widgets is 2 px.

```
button.configure(borderwidth=5) # Change the default.
```

To change the style of the mouse cursor when you hover over a particular widget:

```
button.configure(cursor=‘cross’) #https://www.tcl.tk/man/tcl8.6/TkCmd/cursors.htm
```

Self-study (relief style)

Specifying the styling options at each widget level may be cumbersome to do so individually for each widget (disadvantages).

- It mixes logic and presentation into one file, making the code bulky and difficult to manage.
- Any change in styling has to be applied to each widget individually.
- It violates the “don't repeat yourself (DRY)” principle of effective coding, as you keep specifying the same style for a large number of widgets.

Tkinter now offers the external option database. This is just a text file where you can specify common styling options.

A typical option database text file looks like this:

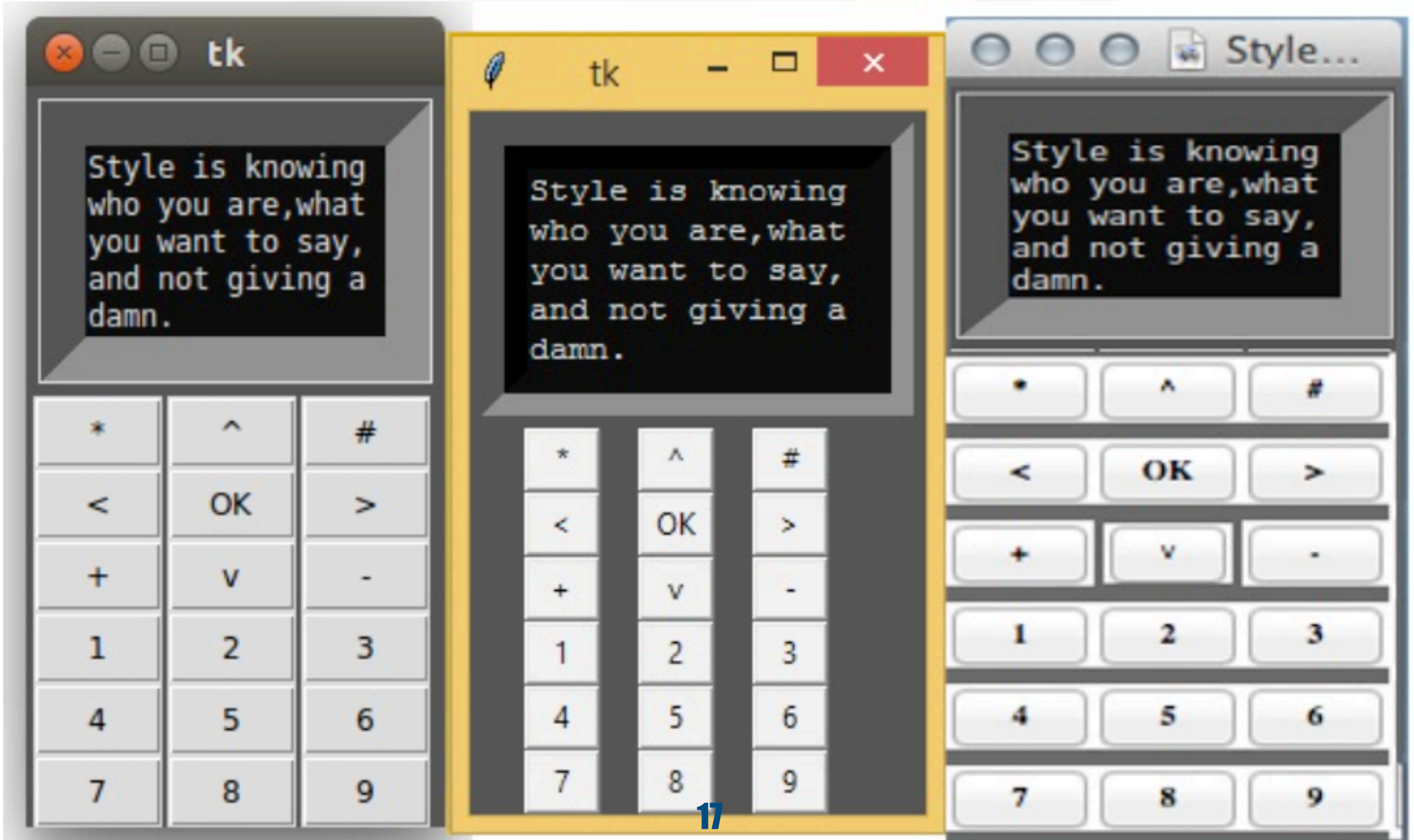
```
*background: AntiqueWhite1
*Text*background: #454545
*Button*foreground: gray55
*Button*relief: raised
*Button*width: 3
```

These entries are placed in an external text (.txt) like ‘optionDB.txt’ file. To apply use :

```
root.option_readfile(‘optionDB.txt’) #Review code 1.12 a demonstration of Tkinter styling.
```

Why?

Lab work! run the code and see which one fits your system!



Display variations between different operating systems such as Ubuntu, Windows, and Mac respectively, as shown in the following screenshot. This is due to differences in the rendering engines of different operating systems

Page 89 Questions

1. What is a root window?

In Tkinter, the root window is the main window of a GUI application. It serves as the container for all other widgets and elements in the user interface. The root window is created using the Tk() constructor.

2. What is the main loop?

The main loop, often referred to as the event loop, is a continuous loop that listens for events (such as button clicks, key presses, etc.) and responds to them. In Tkinter, the main loop is started using the mainloop() method on the root window. It keeps the GUI responsive and handles user interactions.

3. How do you create a root window?

```
import tkinter as tk
root = tk.Tk()
```

4. What are widgets? How do you create widgets in Tkinter?

Widgets are the building blocks of a GUI. They can be buttons, labels, entry fields, etc. In Tkinter, widgets are created using various constructor functions. For example,

button = tk.Button(root, text="Click me”)

5. Can you list or identify all available widgets in Tkinter?

Some common widgets in Tkinter include Button, Label, Entry, Text, Canvas, Checkbutton, Radiobutton, Scale, Listbox, Menu, Frame, and more.

6. What are geometry managers used for?

Geometry managers in Tkinter are used to organize and arrange widgets within a container (e.g., the root window or a frame). They control the placement and sizing of widgets.

7. Can you name all the available geometry managers in Tkinter?

The three main geometry managers in Tkinter are pack(), grid(), and place().

8. What are events in a GUI program?

Events in a GUI program are user actions or occurrences, such as button clicks, key presses, mouse movements, etc. Tkinter provides a mechanism to bind functions (callbacks) to these events.

9. What are callbacks? How are callbacks different from regular functions?

Callbacks in Tkinter are functions that are executed in response to a specific event, such as a button click. They are different from regular functions because they are bound to events and triggered when the associated event occurs.

10. How do you apply callbacks to an event?

Callbacks are applied to events using the bind() method. For example, to bind a function callback_function to a button click event:

```
button = tk.Button(root, text="Click me")
button.bind("<Button-1>", callback_function)
```

11. How do you style widgets using styling options?

Widgets in Tkinter can be styled using various configuration options. For example, to change the background color of a button:

python button = tk.Button(root, text="Click me", bg=“blue")

12. What are the common configuration options for the root window?

Some common configuration options for the root window include bg (background color), geometry (initial size and position), title (window title), resizable (whether the window can be resized), etc.

python root = tk.Tk() root.title("My App") root.geometry("400x300") root.configure(bg="white")

Thank You!