

## LECTURE 22 — TKINTER

### 41.1 Last Four Lectures

Selection of advanced topics:

- Lecture 22: Graphical user interfaces
- Lecture 23: Recursion
- Lecture 24: Functional programming
- Lecture 25: Course summary; intro to C++

### 41.2 Overview of Today's Lecture

- We will learn about event driven programming
- We will discuss graphical user interfaces (GUI)
- We will see how to use TkInter implementation for Python to develop GUIs

### 41.3 Graphical User Interface

- All our programs so far had a terminal based input/output through Wing. This is the simplest type of user interface: enter some text and get some output.
- In real life, we rarely use such programs. All Web interfaces and mobile apps involve elements like buttons, text input fields, radio buttons.

This type of a user interface is called a **graphical user interface** or GUI.

- Designing a GUI is not a simple matter. It requires one to understand effective communication of information through visual interfaces. It is highly recommend you take a class on Visual Communication or Human Computer Interaction to learn more about these.
- Many GUI packages will have some default settings that are meant to help you design interfaces that are at least not awful! So, make sure you do not override them until you know what you are doing.

### 41.4 Event Driven Programming

- A graphical user interface is designed to do something in response to an event.

- Example events are:
  - Button push (most popular so far)
  - Selection from a list of options
  - Mouse movement
- The interface usually runs within a main window (often called root) and listens continuously for events it knows how to process.
  - Hence, a GUI is always in an infinite loop.
  - When an event happens, it executes the function corresponding to the event and continues to listen.
  - To finish the GUI loop, often an explicit termination command has to happen.
- TkInter is a Python module implementing the Tk toolkit for building graphical user interfaces. Other languages also have Tk implementations.
- **Note:** The most important distinction between regular and event based programming is that the action corresponding to an event gets executed in parallel with the rest of the program, it does not block the execution of functions in response to other events.

## 41.5 TkInter First Program

- Let's see a very basic program.

```
from tkinter import * ### (1)

root = Tk()           ### (2)
root.mainloop()       ### (3)
print("Hello")        ### (4)
```

1. It is common to import everything from Tkinter to make it easy to write GUI programs. Compare this with the following program that does the same exact thing but requires using Tkinter for all calls to the module.

```
import tkinter
root = tkinter.Tk()
root.mainloop()
print("Hello")
```

2. We create the main window that the whole application will run in. This main window has many properties already: it can be resized and moved like any window on your computer.
3. We tell the main window to run in an infinite loop, until it is explicitly terminated. In this case, you need to explicitly click on the upper left corner to close this application. This generates a “destroy” event, which terminates the main window.
4. To understand the infinite loop notice that `print "Hello"` command is not executed until the window terminates and the main loop ends.

## 41.6 Widgets and Containers

- GUIs involve two main types of objects: widgets and containers

- Widgets are objects that have a function and are visible. These are the main elements of an interface. We will see the following main widgets:
  - Buttons: can do something when clicked
  - Canvas: you can draw shapes or put shape in them
- There are many other widgets like radio boxes, check boxes, menu of items, etc. You can learn these as we get comfortable with these two.
- Containers are invisible, but they allow us to group widgets and arrange them in visual groups. We will see only one type of container in this lecture:
  - Frames: can contain any of the widgets of the above type
- Containers' size will grow and shrink to fit the objects contained within. Containers can contain other containers as well.

## 41.7 Example use of containers and buttons

- Let's create a first program to create four buttons, put two pairs in one frame, the second pairs in another frame, and then arrange these two frames on top of each other.

```
from tkinter import *

root = Tk()
main_frame = Frame(root)
main_frame.pack()

top_frame = Frame(main_frame)
top_frame.pack(side=TOP)

bottom_frame = Frame(main_frame)
bottom_frame.pack(side=BOTTOM)

button1 = Button(top_frame, text="Top 1")
button1.pack(side=LEFT)
button2 = Button(top_frame, text="Top 2")
button2.pack(side=RIGHT)

button3 = Button(bottom_frame, text="Bottom 1")
button3.pack(side=LEFT)
button4 = Button(bottom_frame, text="Bottom 2")
button4.pack(side=RIGHT)

root.mainloop()
```

- In the above example, the two frames for containing buttons are organized inside another frame.
- The organization between widgets is achieved by a strict hierarchy. Each widget or container has a parent:

GUI Element	Parent
root	None
main_frame	root
top_frame	main_frame
bottom_frame	main_frame
button1	top_frame
button2	top_frame
button3	bottom_frame
button4	bottom_frame

- Whenever you create a frame, button, canvas, the first argument is the parent element.
  - Parent element tells us the placement of the widget, e.g. button1 is placed to the left of top\_frame and top\_frame is to the top of main\_frame and main\_frame is within the main window of root.
  - Parent container's size is determined as a function of the sizes of its children. If nothing was put in a container, its size may be zero.
- Pack will make the interface element a part of the GUI, creating them is not enough. You must attach them to the interface with packing.
- By changing the arguments to pack() we can alter the placement of the buttons and the frames.

## 41.8 Running tkinter programs using classes

- Already our programs are becoming long and so far they don't really do anything yet! What we really want is to have programs accomplish tasks when a button is pressed (for example).
- As it is generally tricky to pass variables between different Tkinter events, we often use classes to implement the main Tk applications.

The member attributes will help us pass values between events.

- We will first rewrite the above program using classes as shown below.

```
from tkinter import *

class MyApp(object):
    def __init__(self, parent):
        self.main_frame = Frame(parent)
        self.main_frame.pack()

        self.top_frame = Frame(self.main_frame)
        self.top_frame.pack(side=TOP)
        self.bottom_frame = Frame(self.main_frame)
        self.bottom_frame.pack(side=BOTTOM)

        self.button1 = Button(self.top_frame, text="Top 1")
        self.button1.pack(side=LEFT)
        self.button2 = Button(self.top_frame, text="Top 2")
        self.button2.pack(side=RIGHT)
        self.button3 = Button(self.bottom_frame, text="Bottom 1")
        self.button3.pack(side=LEFT)
        self.button4 = Button(self.bottom_frame, text="Bottom 2")
        self.button4.pack(side=RIGHT)

if __name__ == "__main__":
    root = Tk()
```

```
myapp = MyApp(root)
root.mainloop()
```

- This is the main idea: Any object outside of root is defined within a class, using root as the parent.
- We can now add functionality to this class that will modify the widgets.

## 41.9 Button Click

- You track many events in a GUI, but we will not see them (such as when a specific mouse button, left or right, is clicked, when it is released, when the cursor is moved, when something is typed on the keyboard, etc.)  
(Note: The distinction between the widget that is the GUI button, and the physical button on your mouse.)
- Command attribute of *Button* allows you to associate a function name with a button click (in GUI terms this is a combination of two events: left button click and its release).

See the following example:

```
self.button = Button(self.bottom_frame, text="Quit", command=self.terminate_program)
```

- The button we generate in this example has a function binding: when it is clicked, it will call the function called *self.terminate\_program*.

## 41.10 Example Program with a Button Click

- This simple program terminates the program by completely destroying the root window.

```
from tkinter import *

class MyApp(object):
    def __init__(self, parent):
        self.parent = parent
        self.main_frame = Frame(parent)
        self.main_frame.pack()
        self.button = Button(self.main_frame, text="Quit", command=self.terminate_program)
        self.button.configure(width=12, padx="4m", pady="4m")
        self.button.pack()

    def terminate_program(self):
        self.parent.destroy()

if __name__ == "__main__":
    root = Tk()
    myapp = MyApp(root)
    root.mainloop()
```

- Here we see the use of classes for two important functions:
  - Keeping track of the root as the parent of all the GUI elements.
  - Defining a function that can access the parent object attached to *self* and destroy it by ending the application.
- We also saw that you can control the size of the button by configuring its width. Padding on x and y allows us to control how the button is placed inside the frame.

## 41.11 Canvas Widget

- This is the last widget we will see today. Canvas is a blank area. You can continuously draw things in it or put text in it. Here are some basic operations.
- Create canvas:

```
canvas = Canvas(parent, height=200, width=200)
canvas.pack()
```

- Draw something in a canvas (an oval within the given box coordinates):

```
canvas.create_oval((40,40,80,80))
```

- Remember, like Image objects, (0,0) represents the top left corner.
- When you draw multiple things in a canvas, it may not immediately show what you draw and a small delay may be noticed. You can force the canvas to show you what you draw with:

```
canvas.update()
```

- You can also draw lines, rectangles and text. See more on:

<http://effbot.org/tkinterbook/canvas.htm>

## 41.12 Timing of GUI Events

- Remember, GUI events are triggered when you click a button (or other events you watch in more sophisticated systems).
- They execute in parallel. If the processing of an event takes some time, it may overlap with another event.
- Sometimes drawing events are too fast for the eye to see the progress. You can incorporate some delays to make this more pleasing to the eye.

```
canvas.after(waittime)
```

which will add a small delay *waittime* to the program.

- Note that *after* and *update* functions apply to many other GUI elements. Mastering them will help you get the exact timing effect from your interface.

## 41.13 Final Program

- Here is a final program to put together a lot of the things we learned. It features a private class method (*new\_button*) that is only accessible to the *\_\_init\_\_* function. It shows how functions can be used to automate repetitive tasks.

```
from tkinter import *

class MyApp(object):
    def __init__(self, parent):
        ## This method is internal to the initializer method
        ## and is used for creating buttons. It shortens the program code
        def new_button(parent, cmd, buttontext, packlocation):
```

```

        button = Button(parent, command=cmd)
        button.configure(text=button_text)
        button.configure(width=button_width,
                          padx=button_padx, pady=button_pady )
        button.pack(side=packlocation)
        return button

#----- constants for controlling layout -----
button_width = 10
button_padx = "2m"
button_pady = "1m"
buttons_frame_padx = "3m"
buttons_frame_pady = "2m"
buttons_frame_ipadx = "3m"
buttons_frame_ipady = "1m"
# ----- end constants -----

#-----variables for controlling the function-----
self.canvas_dimension = 600 ##Canvas will be a square
self.wait_time = 8
self.repetitions = 2
#-----end of variables-----

self.myParent = parent
self.main_frame = Frame(parent)
self.main_frame.pack ()

## Two frames inside the main frame, one for the canvas
## on top and the second one for buttons in the bottom
self.draw_frame = Frame(self.main_frame)
self.draw_frame.pack(side=TOP)

self.info_canvas = Canvas(self.draw_frame, height=20,
                           width=self.canvas_dimension)
self.info_canvas.pack(side=TOP)
self.text_area = self.info_canvas.create_text(10,10,anchor='nw')
self.info_canvas.itemconfigure(self.text_area,text="#circles = {:d}".format(self.
--repetitions))

self.main_canvas = Canvas(self.draw_frame, \
                           height=self.canvas_dimension,
                           width=self.canvas_dimension)
self.main_canvas.pack()

self.button_frame = Frame(self.main_frame)
self.button_frame.pack(side=BOTTOM)

self.draw_button = new_button(self.button_frame,self.draw, 'Draw', LEFT)
self.clear_button = new_button(self.button_frame,self.clear, 'Clear', LEFT)
self.increase_button = new_button(self.button_frame,self.increase, 'Increase', LEFT)
self.reduce_button = new_button(self.button_frame,self.reduce, 'Reduce', LEFT)
self.quit_button = new_button(self.button_frame,self.quit, 'Quit', RIGHT)

def clear(self):
    self.main_canvas.delete("all")

def reduce(self):
    if self.repetitions > 1:

```

```

        self.repetitions //= 2
    self.put_info()

    def increase(self):
        if self.repetitions < 200:
            self.repetitions *= 2
            self.put_info()

    def put_info(self):
        ## Change the text field in the canvas
        self.info_canvas.itemconfigure(self.text_area, text="#circles = {:d}".format(self.
-repetitions))

    def draw(self):
        boundary_offset = 2
        max_radius = (self.canvas_dimension - 2*boundary_offset) // 2
        xc = self.canvas_dimension//2 + boundary_offset
        r = max_radius/self.repetitions
        inc = r
        for i in range(self.repetitions):
            self.main_canvas.create_oval((xc-r, xc-r, xc+r, xc+r))
            r += inc
            self.main_canvas.update() # Actually refresh the drawing on the canvas.
            # Pause execution. This allows the eye to catch up
            self.main_canvas.after(self.wait_time)

    def quit(self):
        self.myParent.destroy()

if __name__ == "__main__":
    root = Tk()
    root.title("Drawing a circle") ##Give a title to the program
    myapp = MyApp(root)
    root.mainloop()

```

## 41.14 Summary

- Graphical user interfaces are event driven. You need to write functions to initialize the interface and change the interface when events happen.
- The most common events are button clicks, but many others are possible.
- Most GUI elements are visible and are called widgets.
- Widgets are placed in invisible containers like frames to group them together.
- Many containers have built-in methods for placing multiple widgets in a way that is pleasing to the eye.
- When building an interface, some of the challenges involve making the interface easy and intuitive to use. Many communications classes concentrate on these issues.