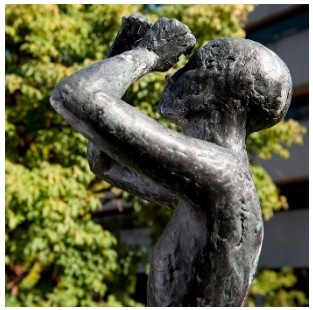

Lecture 16 – tkinter

CS2513

Cathal Hoare

A TRADITION OF INDEPENDENT THINKING

UCC

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Lecture Contents

More TKInter

Parts of an Interface

- The basic components in the library are:
  - Basic Frame
  - Widgets (such as buttons, labels, lists, etc)
  - Layouts
  - Event handlers and event loop

- To implement, we must:
  - Import components
  - Add components (widgets) into a layout
  - Add event handlers for widgets
  - Enter an event loop where we depart from the normal flow of control and enter an event driven model

Event Driven Program

- **Event-driven programming** is a programming paradigm in which the flow of the program is determined by events such as user actions (mouse clicks, key presses);
- Events are generated by the operating system and passed to the running application where they are translated into object instances (Event).
  - Here they are used to trigger event handlers where the event can be passed and values about the event accessed.

Creating a Frame

```
import tkinter as tk                #Add the import


root = tk.Tk()                      #Create a frame object
root.geometry('400x400')            #Simple formatting

root.mainloop()                     #Enter the event loop
```

- We create the frame and configure it
  - This creates a frame that can contain content.
  - Even with just this call we have a frame that can be resized, closed, maximized, etc. It also has a title.
- The event loop is entered with the mainloop call.
  - At this point, the programs flow of control becomes dependent on user actions rather than executing sequentially.

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Adding Components

```
import tkinter as tk


root = tk.Tk()
root.geometry('400x400')

L1 = tk.Label(root, text='Some text’) # Adds a label to root
                                     # initializes with some text
L1.pack()                            # Simple layout to add to frame

btn = tk.Button(root, text="OK")     # Adds a button
btn.pack()

root.mainloop()
```

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Event Handler

```
import tkinter as tk

def onclick():
    L1.config(text = "Modified text") # The function that is called
                                      # when the button is pressed.
                                      # In this case updating the label


root = tk.Tk()
root.geometry('400x400')

L1 = tk.Label(root, text='Some text')
L1.pack()

btn = tk.Button(root, text="OK", command=onclick)
                    # We modify the button to name the method
                    # called when the button is pressed.
btn.pack()

root.mainloop()
```

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Event Handler

- Note we pass the name of the function to the Button.
  - We are operating within the framework. The Button implementation (TkInter's code) needs to know the name of the function to call. We pass that name and also provide an implementation of the function so that it can be called.
  - In this way we provide meaning to the button – until we provided that name, the button was a generic TKInter button – now it is a button that executes our code.
  - Note: we are also working within the bounds set by the framework – we are implementing our code in a way dictated by the framework and the objects it implements.

Other forms of Input

- As well as clicks on controls we can also handle key strokes and mouse clicks on the overall application window.
- For a mouse click - we can bind a mouse button to some action.

Function we bind to

Where we bind to the interface component

```
import tkinter as tk

def mouse_function(event):

    myCanvas.create_oval(event.x, event.y, event.x + 20, event.y + 20, fill='red')

root = tk.Tk()
myCanvas = tk.Canvas(root, bg="white", height=300, width=300)
myCanvas.bind("<Button-1>", mouse_function)

myCanvas.pack()
root.mainloop()
```

Organizing our code in an OO Way

- When we develop a TKInter App there are two phases:
  - Setting up the interface, laying it out and creating actions to execute
  - Providing actions to execute

- This maps to a simple class definition:
  - Setup can be done wither entirely or by method calls from the constructor
  - The actions can be added as methods to the class.


Next Time:
TKInter