CIS 112

Intro to Programming Using Python

Module 4 Part 1

Agenda for the Day!

- GUI programming
  - Basics
  - Events,
  - Widgets

Overview of GUI Programming in Python

Taking your programming to the next level by incorporating GUIs!

- Most of you have written code and run it in a command-line terminal or an IDE
- The code produces an output based on what you expect out of it either on the terminal or on the IDE itself.
- However, what if you want your system to have a fancy looking user-interface or maybe your application (use-case) requires you to have a GUI.
- GUI is nothing but a desktop app that provides you with an interface that helps you to interact with the computers and enriches your experience of giving a command (command-line input) to your code.
- GUI's are critical for making user-friendly applications, and allow your applications to come to life!

So how do we do GUIs in Python?

- So how does one do GUI programming in Python?
- Turns out there are a lot of ways!
  - Some some libraries include:
    - PyQT
    - Kivy
    - Jython
    - WxPython
    - PyGUI
- And finally, the framework that we'll be utilizing in this module: Tkinter!
  - Tkinter commonly comes bundled with Python and is Python's standard GUI framework.
  - It is famous for its simplicity and graphical user interface. It is open-source and available under the Python License.

TKInter Basics

New Paradigm: Event-driven programming

- Event-driven programming
- Anything that happens in a user interface is an event. We say that an **event is fired** whenever the **user does something - for example, clicks on a button or types a keyboard shortcut**.
- Some events could also be triggered by occurrences which are not controlled by the user - for example, a background task might complete, or a network connection might be established or lost.
- Our application needs to monitor, or **listen for**, all the events that we find interesting, and respond to them in some way if they occur.
- To do this, we usually **associate certain functions with particular events**. We call a function which performs an action in response to an event an **event handler** - we **bind handlers to events**.

TKInter Elements

- **tkinter** provides us with a variety of common GUI elements which we can use to build our interface - such as **buttons**, **menus** and various kinds of **entry fields** and **display areas**.
- We call these elements **widgets**.
- We generally construct a tree of widgets for our GUI - each widget will have a parent widget, all the way up to the **root window of our application**.
  - For example, a button or a text field needs to be inside some kind of containing window.
- The widget classes provide us with a lot of default functionality.
  - They have methods for configuring the GUI's appearance - for example, arranging the elements according to some kind of layout - and for handling various kinds of user-driven events.

TKInter Steps

Basic Steps to a TKInter App

---

1. **Importing the Library:** To use Tkinter, you start by importing it into your Python script:

```
import tkinter as tk
```

2. **Creating a Main Window:** The Tk() class creates the main window of the application, also known as the root window.

```
root = tk.Tk()
```

Basic Steps (cont.)

---

3. **Adding Widgets:** As we noted Tkinter provides various widgets (such as buttons, labels, entry fields, etc.) that you can add to your GUI.

You create these widgets and then place them within the main window or other container widgets like frames or canvases.

```
label = tk.Label(root, text="Hello, Tkinter!")
button = tk.Button(root, text="Click me!")
```

Basic Steps (cont.)

4. Layout Management: Tkinter provides several ways to organize and manage the layout of widgets within a window. The two main layout managers are:

- **Pack**: The pack() method organizes widgets in a block, either horizontally or vertically.

```
label.pack()
button.pack()
```

- **Grid**: The grid() method arranges widgets in a grid-like structure, with rows and columns.

```
label.grid(row=0, column=0)
button.grid(row=1, column=0)
```

Basic Steps (still!)

---

5. **Event Handling**: Tkinter allows you to bind functions (event handlers) to specific events, such as button clicks or key presses. This allows your GUI to respond to user interactions.

```
def button_click():
    print("Button clicked!")

button = tk.Button(root, text="Click me!", command=button_click)
```

6. **Running the Application:** After setting up the GUI components, you start the Tkinter event loop by calling the **mainloop()** method on the root window. This method listens for events such as button clicks and updates the GUI accordingly.

```
root.mainloop()
```

Sample App

Example App

...