**GRAPHICAL USER INTERFACE USING PYTHON-TKINTER**

Hello Guys,

I am going to walk you through the basics of a GUI development using Python-Tkinter module. I have written this for Python 2.7. You can use it for higher versions with very minimal change. I have given very simple examples and I will highlight the key things to be understood in order to effectively create a GUI. I am sure this can be a great starting point to learn python where you can create something useful and visualize which will improve your interest in coding.

**1. INTRODUCTION**

The first thing to think about when we develop a GUI is to visualize how a GUI looks like! Think of any application that you use and what you see when you click on it is a GUI. Every GUI has elements like Titlebar, Menubar, Toolbar, buttons, radiobuttons, checkboxes etc etc. But every element that we talked about before is actually attached to a common platform called a “Frame”. This is as simple as “create a frame and put all the stuff that you want it to display!”

The way a GUI is developed in Tkinter is using “widgets” and you go on adding the widgets in the code as you develop the GUI. I will list down the widgets that we commonly use. Each widget is defined as a class in Tkinter. So you just create an object of the widget class to use and you can access all other ‘configuration options’ and ‘methods’ that go along with widget.

All widgets are properly documented and can easily be searched which will provide information on all the “configuration options” that the widget has and how to use them. Everything is pretty straight-forward, exactly same process and easy!

“Welcome to the Python World”

**2. WIDGETS**

|  |  |
| --- | --- |
| Frame | As the name suggests, this is the ‘frame’ where you attach widgets to (think of a photo frame where you can stick photos, stickers etc) |
| Menu | This is used to create a menubar |
| Label | Creates a Label that you can use to display text or anything on the frame |
| Entry | To get user- input (Think of ‘log-in’ & ‘Password’ entries) |
| Radiobutton | Creates radiobutton option to get user-selection |
| Button | Creates a normal button to which you can assign a command to do when the button is clicked |
| PanedWindow | Creates a paned window where you can add different widgets in specific panes (Think of having Left pane and right pane in a normal windows application) |
| Treeview(ttk) \*\* | Creates a treeview (Column view) to navigate through different frames |

\*\*Most of the widgets are defined in as ‘tk’ widgets and some of them are developed as ‘ttk’ widget which is an extension to Tkinter modules. If it’s mentioned as ttk, it means its avaible only in ttk class.

**3. CREATING A WINDOW:**

The first thing to do when creating a GUI in Tkinter is to create the “**Root Window**”. Every Tkinter Application must have only one root window. We can create a root window for our application by instantiating Tk() object like given below. You can think of a window like a normal MS Word window with a title bar, menubar, toolbar, workspace to type (i.e. a Frame) and all other options to work with as a ‘whole’. When you close this root window, the application will be closed!

**Note:** Instantiating **Tk** does more than creating a window, it initializes the entire Tkinter framework. It actually starts up a hidden ‘tcl’ interpreter which does the actual work of managing widgets. Tkinter is just a python wrapper around this interpreter.

**from** Tkinter **import** \*

root = Tk()  
mainloop()

Let us go line by line:

**from** Tkinter **import** \*

The first line is how we import a module into your python script. It means import all sub-modules of Tkinter class.

**Note:** If the module comes with python base package like Tkinter you can directly import it. If you want to use modules like ‘numpy’ which doesn’t come with base package, it will raise an error. You need to install the module using pip like explained before.

root = Tk()

Here we are instantiating (creating an object) of Tk() class which will create the Root window for our application that we talked about above and this will manage all the other widgets that we are going to create for our application.

mainloop()

Mainloop() method which is kind of an infinite loop that will display the frame until the event is destroyed. In this case, until you click the ‘close’ button. You can think of **system(“pause”)** in C++ but it is not exactly the same.

**3.1 OOPS!**

We have understood what a root window is and instantiated it in the simplest way possible. Let’s go the OOP way to create a root window and customize the window of our application.

from Tkinter import \*  
  
class Root\_Window (Tk):  
 def \_init\_(self,\*args,\*\*kwargs):  
 Tk.\_init\_(self,\*args,\*\*kwargs)  
  
 self.geometry("1280x720")  
 self.title("My First GUI”)  
  
app= Root\_Window ()  
app.mainloop()

class Root\_Window (Tk):

Here we are creating the class called “Root\_Window” and we are inheriting from the “Tk” class which will create the Root window for our application. Remember there should be only one root window for an application and it is the first step to start with the GUI.

def \_init\_(self,\*args,\*\*kwargs):

This function at the start of the class is called “**initialization function**”. It is like a “Constructor” in C++ which will instantiates the class when an object of this class is created. And all the variables and default methods that you want to start when calling an object of this class should go under this. That means, whatever you want to display once the user opens the application should be declared under this function.

**Note:** If you want to define different functions for this class separately but still want to start the function at the start of the class, you can call the function from \_init\_ function so that once the object is created, the \_init\_ function calls the function that you want.

For knowing what \*args and \*\*kwargs mean, go through this link which explains it perfectly:  
<http://pythontips.com/2013/08/04/args-and-kwargs-in-python-explained/>

Tk.\_init\_(self,\*args,\*\*kwargs)

Here, we are calling the initialization function of ‘Tk’ class from which we inherited the Root\_window class. This will instantiate the ‘Tk’ class which in turn creates the root window for our application.

self.geometry("1280x720") self.title("iAircraft\_GUI")

To understand about ‘self’ in python class, go through this link which is very key to understanding how class in python works: <http://pythontips.com/2013/08/07/the-self-variable-in-python-explained/>

In this code, we are setting the ‘Title’ and ‘Geometry’ of the root window that we are creating. Remember this means it is for the entire GUI application that we are creating.

5.

app = Root\_window()  
app.mainloop()

Here we are creating an object for the Root\_window class and calling the mainloop() method to show the window till the event is destroyed.

**4. ARCHITECTURE OF GUI (CREATING MULTIPLE FRAMES):**

Now we have the root window of our GUI ready. So, let us start building it using the widgets listed above to do some useful stuff!

First, I will explain the method to switch between different frames. A GUI will obviously contain multiple frames displaying different content. So let us see how to create multiple frames in a way that we can switch between them and show the frame that user wants to see.

We will create a **“Container”** which is a frame itself and arrange all other frames that we want to create in this container like a **stack of frames**. You can imagine something like below:

|  |
| --- |
| Frame 4 |
| Frame 3 |
| Frame 2 |
| Frame 1 |

CONTAINER

Once we create the stack of the frames like this, we can “raise” the frame we want to the top of the stack so that the user can see that particular frame and other frames are hidden behind.

This can be done using **“tkraise”** method to bring frames to the top of the stack.

First we will create the top container for containing other frames and pack that frame in our root window and then create a Menubar that will remain attached to the container.

**4.1 CREATING TOP CONTAINER:**

The following code creates a container which is a frame itself.

from Tkinter import \*  
  
class Root\_Window (Tk):  
 def \_init\_(self,\*args,\*\*kwargs):  
 Tk.\_init\_(self,\*args,\*\*kwargs)  
 self.geometry("1280x720")  
 self.title("MY FIRST GUI")

Top\_container=Frame(self)

Top\_container.pack(side="top", fill="both", expand=True)  
 Top\_container.grid\_rowconfigure(0,weight=1)  
 Top\_container.grid\_columnconfigure(0,weight=1)

app= Root\_Window ()  
app.mainloop()

One important thing to learn when we start creating widgets to attach to the root window or a frame, is to learn about different “**Geometry Managers**” in python on how to **arrange** these widgets on the frames.

Kindly go through this link in detail to perfectly learn about geometry managers which is essential to arrange widgets in the way we want: <http://effbot.org/zone/tkinter-geometry.htm>

**IMPORTANT NOTE:** Only the root window will be shown once the Tk object is instantiated. All other widgets like frames, buttons, labels have to be arranged using the geometry manager. Unless and until, you arrange (pack) the widgets to a frame or any other window (container), the widget **won’t be visible.**

Top\_container=Frame(self)

Top\_container.pack(side="top", fill="both", expand=True)  
 Top\_container.grid\_rowconfigure(0,weight=1)  
 Top\_container.grid\_columnconfigure(0,weight=1)

The first line is creating a frame called “Top\_container” on which we are going to stack other frames. Here you can see that we have packed the frame to the root window using Pack manager. It’s important to know that this top container is attached to the root window that we created initially and we have pasted all over the root window using the pack manager.

**4.2 CREATING MENUBAR FOR TOP CONTAINER:**

Now we have a top container which will be visible once the user opens the application. It is attached to the root window all over and it is empty right now.

So, let’s add a ‘menubar’ to the root window which we need it to be visible for all the frames that we are going to create.

menubar = Menu(self)

self.config(menu=menubar)  
  
fileMenu =Menu(menubar, tearoff = False)  
menubar.add\_cascade(label="File", menu=fileMenu)  
fileMenu.add\_command(label='New')  
fileMenu.add\_command(label='Open')  
fileMenu.add\_command(label='Save')  
fileMenu.add\_command(label='Exit', command = quit)

The above code is how you can create a Menubar widget in Tkinter. In the first line, we are creating an object of the Menubar class and attaching it to the ‘Root Window’ that we have created before. (It is critical to understand the meaning of ‘self’ in python classes)

Once we add the menubar, we need to create the different options that we want to display in the menu bar. We will add a ‘File’ menu and you can do the same exact thing for all other menu options that you want to create.

fileMenu =Menu(menubar, tearoff = False)  
menubar.add\_cascade(label="File", menu=fileMenu)

Here we are adding it as a ‘cascade’ menu as it is the conventional way of displaying a menu and you can add the options as ‘commands’ to the Menu. These options will be displayed as a drop down box at the top.

**Note:** It is important to note that we have just displayed the menu options and in reality they don’t do any function if the user clicks the option. Hence, we need to add specific functions to define what should be done when the user clicks it.

We will add one functionality for an example and as we go we will learn how to attach specific functions to the widgets like buttons, radiobuttons etc. that will define that widget’s functionality.

fileMenu.add\_command(label='Exit', command = quit)

In this above line, we have added a functionality to the ‘Exit’ option using ‘command’ parameter. ‘Quit’ is an in-built function in Tkinter which will close the current root window. Hence, you just need to specify that function as the command that should be executed when the user clicks that particular widget. This is fairly simple because we are using a built in function. For all other specific functions, you will need to define a function and call that function using the ‘command’ parameter.

**4.3 CREATING FRAMES:**

Now we have come to the important aspect of creating multiple frames which will display the things that we want the user to see and make selection to navigate between different frames based on their selection.

We will create three frames (a Start Page and two more pages) in the OOP way and we will navigate between them. As it was explained before, we need to attach all the frames to the ‘Top container’ and raise the selected frame to the top above all.

class Start\_Page(Frame):  
 def \_\_init\_\_(self,parent,controller):  
 Frame.\_\_init\_\_(self,parent)  
  
 label =Label(self, text="WELCOME TO MY GUI",font=LARGE\_FONT)  
 label.pack(pady=10,padx=10)  
  
 Radio\_button\_1 = Radiobutton(self,text="GO TO PAGE ONE",

command=lambda: controller.show\_frame(Page\_One))   
 Radio\_button\_1.pack()  
  
 Radio\_button\_2 = Radiobutton(self,text=" GO TO PAGE TWO",

command=lambda: controller.show\_frame(Page\_Two))  
 Radio\_button\_2.pack()

We have created a Frame just like how we created the root window. Create a Class for all the frames that you want to create so that we can call that frame using the object of that class.

We will go ahead and create two more frames for importing input file directly and another to define an aircraft.

class Page\_One(Frame):  
  
 def \_\_init\_\_(self,parent,controller):  
 Frame.\_\_init\_\_(self,parent)  
  
 self.label = Label(self, text="YOU HAVE COME TO PAGE ONE",font=LARGE\_FONT)  
 self.label.pack()

self.button = Button(self,text="Back to Home",

command=lambda: controller.show\_frame(Start\_Page))  
 self.button.pack(side = "bottom")

With the above piece of code, we are creating a frame called Page One with a Label widget saying that and a Back button to go back to Start Page. For now, ignore the command option. We will come to that eventually.

class Page\_Two(Frame):  
  
 def \_\_init\_\_(self,parent,controller):  
 Frame.\_\_init\_\_(self,parent)  
  
 self.label = Label(self, text="YOU HAVE COME TO PAGE TWO",font=LARGE\_FONT)  
 self.label.pack()

self.button = Button(self,text="Back to Home",

command=lambda: controller.show\_frame(Start\_Page))  
 self.button.pack(side = "bottom")

We are creating another frame called Page Two with a label and a button.

1. Top Container attached to root window
2. Start\_Page frame
3. Page\_one frame
4. Page\_two frame

**4.4 SWITCHING BETWEEN FRAMES:**

Now let’s move on to the interesting part. How to switch between them?

We are going to create a ‘Dictionary’ of objects for each of these classes first. We have to remember that we need to attach all frames to the Top container.

And also we need to create a ‘function’ which will raise the selected frame to the top. As this function is defined in ‘Root\_window’ class, we cannot access it from other classes to use the back button to switch back to the start page. Hence we will create the objects for the other frame classes in such a way that we can access the function to switch frame.

class Root\_window(tk.Tk):  
 def \_\_init\_\_(self,\*args,\*\*kwargs):  
 tk.Tk.\_\_init\_\_(self,\*args,\*\*kwargs)  
  
 self.geometry("1280x720")  
 self.title("iAircraft\_GUI")  
  
 Top\_container=Frame(self)  
 Top\_container.pack(side="top", fill="both", expand=True)  
 Top\_container.grid\_rowconfigure(0,weight=1)  
 Top\_container.grid\_columnconfigure(0,weight=1)  
  
  *#Creating a dictionary of objects*

self.frames = {}   
   
 for F in (Start\_Page,Page\_One,Page\_Two):  
  
 frame = F(Top\_container,self)

*#Objects for each frame class created with Top container as first parent object and also passing object of Root Window class so that all other frame classes can access the functions defined here using that object*

self.frames[F] = frame  
 frame.grid(row=0, column=0, sticky="nsew")  
  
 #Bring the Start Page initially  
 self.show\_frame(Start\_Page)  
  
 *#Method to pull different frames to the top*  
 def show\_frame(self,cont):  
 frame=self.frames[cont]  
 frame.tkraise()

The ‘for loop’ will create an object for each frame class and store it in the dictionary. The function “show\_frame” will pull up the frame that is being passed as an argument.

We have created a GUI with three Frames where you can display whatever stuff you want by adding different widgets and packing them onto those frames.

Now, let us talk about the ‘command’ option of Button widget. Here we have ‘attached’ a function to specify what the widget must do when the user clicks on it. This is how you make widgets to do different things you want.

It is important to learn about what ‘lambda’ function does. You can read about it here: <http://effbot.org/zone/tkinter-callbacks.htm>

Basically it allows us to pass arguments to the function that is being called when we click that button. Here we need to pass the frame class as an argument to show\_frame function which is exactly been done. So when the user clicks the radiobutton, that particular frame will be brought up. In this manner, you can create infinite frames and switch between them with ease.

Now we have created our own GUI within minutes! How awesome is that! We can create a GUI for whatever application we write and provide an intuitive way to interact with our application!

We have touched upon the basics that you need to know for developing a GUI. You can find the source code of the example I used to explain these things in the same Github folder.

I will keep adding up more examples of GUI which you can try out. Be sure to try these things out and be creative! You can become a Front End developer in the near future!

***“Look at the sky. We are not alone. The whole universe is friendly to us and conspires only to give the best to those who dream and work”***

***- Dr.A.P.J.Abdul Kalam***

With Love,  
Mugesh