Brief Intro to Tkinter UI

This document provides some basic concepts about Python's Tkinter package for GUI creation.

Tkinter is Python's de-facto way to create Graphical User Interfaces (GUIs) using the Tk GUI toolkit. Understanding all the mechanisms is beyond the scope of this document (and course) but here we cover the basic concepts to get you started and apply to other GUI frameworks. These concepts include: installing Tkinter, creating widgets, putting things together, and making the interface interactive.

Installing Tkinter

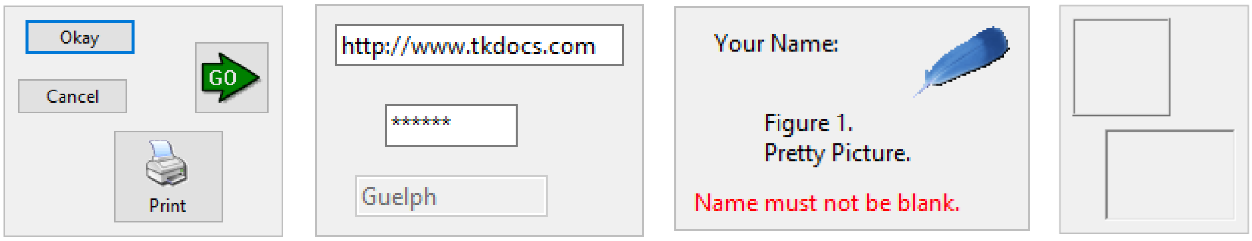
This is the easy part. When you install Python 3.1 or above (using the binary installer) from https://python.org, Tkinter comes with its standard library. You'll have to do some extra work if you want to compile Python yourself, but this is beyond this document.

To actually use Tkinter, you'll have to import the package. The most common way is to do this:

```
from tkinter import * # lets you use everything in tkinter directly
from tkinter import ttk # lets you use some new widgets under the ttk package
```

Creating Widgets

Widgets are UI components that you add to the GUI. Some of them are easily recognizable and interactive like buttons and entries, some are more subtle like labels and frames.



Buttons Entries Labels Frames

Looks might be slightly different across OSs and versions.
Source: https://tkdocs.com/tutorial/widgets.html

The general code to create a widget is to either call the corresponding class creation function from tkinter (just the function name) or from ttk (function with the **ttk.** prefix):

```
button = ttk.Button(<parent>, text="OK", command=<callback>) # a button
entry_text = StringVar() # part of entry creation
entry = ttk.Entry(<parent>, textvariable=entry_text) # an entry
label = ttk.Label(<parent>, text="input:") # a label
frame = ttk.Frame(<parent>) # a frame (usually to hold other widgets)
```

There are different ways to call these functions (using different arguments) to control various characteristics of the widget (e.g., to use image or text on a button/label), partly depending on which widget it is. However, one argument that is necessary in all cases is the **<parent>**, which basically means where does this widget belong to – every widget needs to belong to something, as explained in the next section.

Once your widget is created, you can call its **.configure()** method to customize it further. For example, if later you want to change the text attribute of the button you created previously to "OKAY", you can do this:

```
# the widget called button is created above
button.configure(text="OKAY")
button["text"] = "OKAY" # this is another way
```

Different widgets have different attributes to be customized. Refer to the widget's reference manual for details: https://tcl.tk/man/tcl8.6/TkCmd/ttk_button.htm (go up one level to see that for other widgets). Conversely, you can retrieve the current value of an attribute using the **.cget()** method (some might only be available via the **.winfo_<attributeName>()** method).

Text and Images

Sometimes you might want to access the text attribute of a widget to either set or query it. Tkinter provides a way for your widget to "monitor" a variable so anytime this variable changes the widget's text will also change. This is done by creating and setting a **StringVar** instance:

```
# the widget entry and the StringVar entry_text are created above
print(entry_text) # prints the current text used by entry
entry_text.set("Input text here") # text of entry will be updated
```

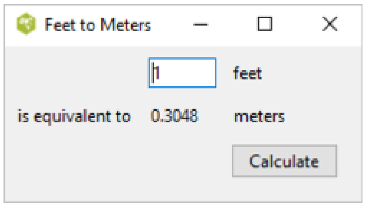
Tkinter uses **PhotoImage** to represent an image file store in your computer. To show it in the UI, a **Canvas** widget is needed to hold it. The following code shows the process of creating a Canvas widget, loading an image file, resizing the widget, and making it hold the image:

```
canvas = Canvas(<parent>, width=640, heigh=480, background="gray75")
my_image = PhotoImage(file="image.png") # load an image file called image.png
canvas.configure(width=my_image.width(), height=my_image.height())
canvas.create_image(0, 0, image=my_image, anchor="nw") # place image at 0, 0
```

You can call **.create_image()** again to hold another PhotoImage. Currently, PhotoImage is limited to only a few image formats including PNG, GIF, and PPM/PNM. You'll need a Tk extension library called Img, or a made-for-Python image library called PIL to accept more formats.

Putting Things Together

Tkinter organizes all widgets in a **widget hierarchy** (also called windows hierarchy), where everything belongs to a single root at the top of the hierarchy, directly or indirectly:

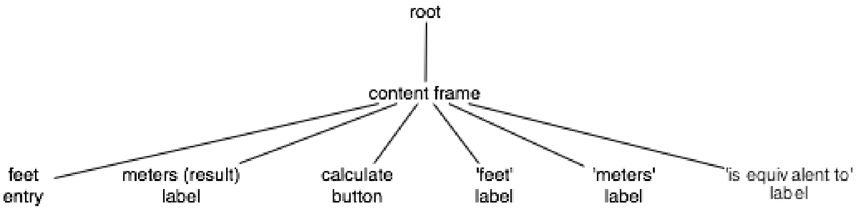


Feet to Meters

1 feet

is equivalent to 0.3048 meters

Calculate



Source: https://tkdocs.com/tutorial/concepts.html

This hierarchy can be arbitrarily deep, depending on how complex your UI is and how you want to organize the widgets. Most UI frameworks use this concept to organize their components.

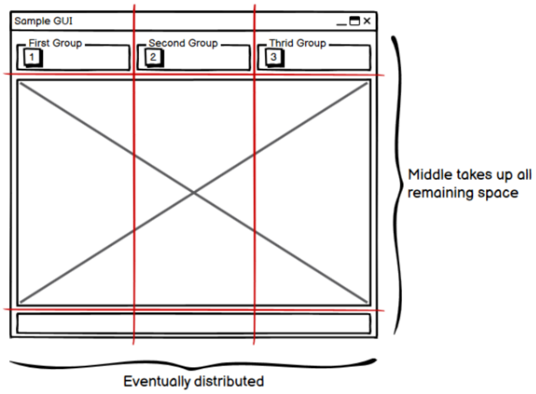
Typically, you would want to put related UI widgets under the same container widget (e.g., a frame). This allows you to organize your code better and even name them in a structured way, (e.g., a frame called "user_input_frame" containing all the widgets used for user input with "user_input" as the prefix of their names). However, this does not tell Tkinter how to place them visually in the interface. This requires a separate step called geometry management.

Geometry Management

Tkinter provides a few geometry managers that figure out exactly where the widgets are going to be placed and how much space each takes given the hierarchy, even when the UI is being resized. One commonly used manger is called pack but it is harder to use. Here we use a more modern and often easier one called **grid**. To use it, you just need to specify 3 things:

- The parent-child relationship between two widgets (typically container-component).
- How child components are distributed spatially (grow/shrink) inside the parent.
- Where is the child placed inside the parent.

For example, suppose you want to create a GUI that has 3 groups of buttons at the top, 1 image in the middle, and 1 status label at the bottom. You can first sketch out the UI using gridlines:



The content area is split into a 3-by-3 grid.

With that you can figure out the widget hierarchy. By convention, we start with a content frame that holds all the widgets in the UI. This content frame is a widget itself and has the "root" as its parent (this will be the first thing you create in your code, some textbooks call it "window").

```
#import all the necessary packages including tkinter and ttk
root = Tk()
root.rowconfigure(0, weight=1) # widget placed at row 0 fills up the space
root.columnconfigure(0, weight=1) # same for widget placed at column 0

# create the content frame
content_frame = ttk.Frame(root, padding="5") # set its parent, add padding
content_frame.grid(row=0, column=0, sticky=N+E+S+W) # place it inside root
content_frame.columnconfigure(0, weight=1) # even distribution across columns
content_frame.columnconfigure(1, weight=1)
content_frame.columnconfigure(2, weight=1)
content_frame.rowconfigure(1, weight=1) # widget at row 1 fills up the space

# create the label frame for the first group, it's a frame with a label
first_lblFrame = ttk.LabelFrame(content_frame, text="First Group")
first_lblFrame.grid(row=0, column=0, sticky=N+E+S+W)
# create a button in this group
first_btn = ttk.Button(first_lblFrame, text="1")
first_btn.grid(row=0, column=0) # place it at 0, 0

# create the label frames for the second and third group...
# write the code yourself :)

# create the canvas and show an image
canvas = Canvas(content_frame, width=640, height=480, background="gray75")
canvas.grid(row=1, column=0, columnspan=3, sticky=N+E+S+W)
my_image = PhotoImage(file="image.png") # load an image file called image.png
canvas.configure(width=my_image.width(), height=my_image.height())
canvas.create_image(0, 0, image=my_image, anchor="nw") # place image at 0, 0

# create the status label together with the monitor variable
status_text = StringVar()
status_text.set("image opened")
status_lbl = ttk.Label(content_frame, textvariable=status_text)
status_lbl.grid(row=2, column=0, columnspan=3, sticky=N+E+S+W)

# now that we have everything, we might want to resize the window
proper_height = first_lblFrame.winfo_height() + int(canvas.cget("height"))
proper_height += status_lbl.winfo_height()
root.geometry(str(canvas.cget("width"))+"x"+str(proper_height))

# need this line at the end for the interactivity
root.mainloop()
```

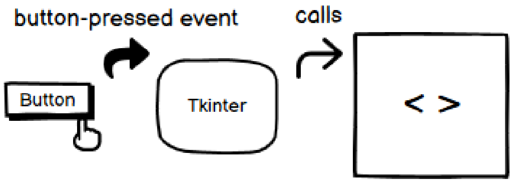
Just remember: state parent, configure spatial distribution (for container widgets), and grid.

Making The Interface Interactive

With the UI properly showing, the last part is to make it interactive. Here we only cover what happens when the user clicks a button (i.e., handling a button-pressed event).

Event-handling Mechanism for Buttons

Tkinter's applications are event-driven, meaning that the application responses to events by calling corresponding functions. When a button is pressed, a button-pressed event is generated and Tkinter looks for function(s) that are designated to handle this event. Your job is to define such function(s) and tell Tkinter to call it when it intercepts a button-pressed event.



For example, suppose you want the status label at the bottom of the sample GUI to print "Hello!" when the button in the first group is pressed. Here is the code you need:

```
# define a function to handle the button-pressed event from first_btn
def handleBtnPressed():
  status_text.set("Hello!")

# rest of the GUI building code... until just before you create the button
first_btn = ttk.Button(first_lblFrame, text="1", command=handleBtnPress)

# rest of the GUI building code... finish with root.mainloop()
```

The code above tells Tkinter to call a function when a button-pressed event is generated (user presses the button). This function is also referred to as the "callback" of the button widget.

There are other kinds of events (e.g., when a keyboard key is pressed, when the mouse moves) and ways to "bind" them to handling functions, which are outside the scope of this document. For more information you can refer to https://tkdocs.com/tutorial/concepts.html#events.

Useful Bits for your GUI

There are some other useful tools offered in Tkinter. One of them is the Dialog Windows, which can be imported like the ttk sub-package for different tasks. For example, import **filedialog** if you want the user to select files or directories, and import **messagebox** if you want to show simple modal alerts to notify the user about something or get their confirmation.

References:

The Tkinter tutorial this document is mostly based on: https://tkdocs.com/index.html
For more examples (and something beyond): https://realpython.com/python-gui-tkinter/