



# Basics of Programming through Python

# GUI in Python

Introduction to Programming

COMP102

Term 3-2022-2023

# Python GUI Programming With Tkinter (TK)



# Lesson Outcomes

- Introduction to Tkinter
- Building Your First Python GUI Application With Tkinter
- Working With Widgets
- Controlling Layout With Geometry Managers
- Making Interactive Applications





# What is Tkinter?

- Tkinter is a [GUI framework](#) that is built into the Python standard library
- Tkinter is a shortcut of “Tk interface”
- **Tkinter Strengths:**
  1. **Cross-platform**, the same code works on [Windows, macOS, and Linux](#)
  2. Visual elements are rendered using native operating system elements
  3. Lightweight and relatively uncomplicated to use compared to other frameworks



# Building Your First Python GUI Application With Tkinter

- The foundational element of a Tkinter GUI is the **window**
- **Windows** are the containers in which all other GUI elements live
- Other GUI elements such as **text boxes, labels, and buttons**, are known as **widgets**
- **Widgets are contained inside of windows**





# Definitions

- **Window:**

A rectangular area somewhere on your display screen

- **Widget:**

The generic term for any of the building blocks that make up an application in a graphical user interface. Examples of widgets: buttons, radio buttons, text fields, frames, and text labels

- **Frame:**

In *Tkinter*, the Frame widget is the basic unit of organization for complex layouts. A frame is a rectangular area that can contain other widgets





# Example: Create a window that contains a single widget

Open Python shell, then the first thing you need to do is to import the Python GUI Tkinter module:

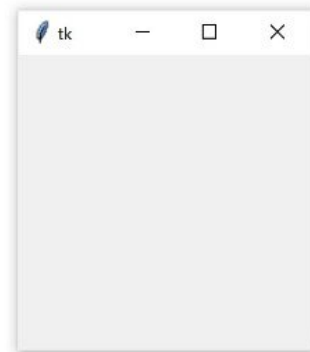
```
>>> import tkinter as tk
```

A **window** is an instance of Tkinter's Tk class

Create a new window and assign it to the variable `window`:

```
>>> window = tk.Tk()
```

Window variable can be any name



(a) Windows

When you execute the above code, a new window pops up on your screen. How it looks depends on your operating system



## Exercise:

- Rename the `title` of the Window to "Welcome":
- Using `title()` method , `window.name.title("")`

Script:

```
from tkinter import *
```

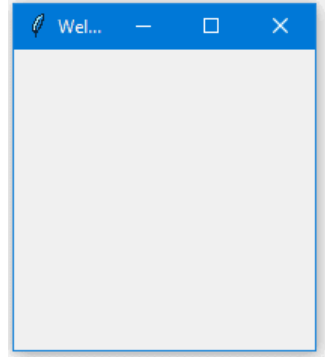
```
window = Tk()
```

```
window.title("Welcome")
```

# in default title will be (tk) if did not used title method

```
window.geometry("200x100")
```

```
window.mainloop()
```



Q: What will happen if execute the following:  
>>> window.mainloop() ?





If tk is not recognized, write the following:

1) In python console

```
>>>import tkinter
```

```
>>>Tkinter._test()
```

2) In terminal

```
pip install tk
```

# Resizable window

**resizable()** method.

This method is used to resize the Tkinter window for Axes (x,y) as arguments by set values **True** , **False**

```
Window_name.resizable(False,False)
```

```
Window_name.resizable(True,True)
```

```
Window_name.resizable(True,False)
```

```
Window_name.resizable(False,True)
```



# Change the dimensions of window

**geometry()** method.

This method is used to set the dimensions of the Tkinter window

```
Window_name.geometry('widthxheight')
```

“ “ or ‘ ‘



`window.mainloop()`: tells Python to run the Tkinter **event loop**. This method listens for events, such as button clicks or keypresses, and blocks any code that comes after it from running until you close the window where you called the method.

**Note:** to close a window:

```
>>> window.destroy()
```

You can also close it manually by clicking the *Close* button



# Working With Widgets

- What are Widgets?

They are the elements through which users interact with the program

- Here are some of the widgets available:

Widget Class	Description
<b>Label</b>	A widget used to <b>display</b> text on the screen
<b>Button</b>	A button that can contain text and can perform an action when clicked
<b>Entry</b>	A text entry widget that allows only <b>a single line of text</b>
<b>Text</b>	A text entry widget that allows <b>multiline text entry</b>
<b>Frame</b>	A rectangular region used to group related widgets or provide padding between widgets

All widgets should be capital for first letter



# Displaying Text and Images With Label Widgets

- **Label** widgets are used to display text or images
- The text displayed by a Label widget can not be edited by the user
- It is for **display** purposes only

```
>>> label = tk.Label(text="Hello, Tkinter")
```

**If you used**  
Import tkinter as tk  
Win=tk.Tk()

Or

```
lb1 =Label(text="Hello, Tkinter")
```

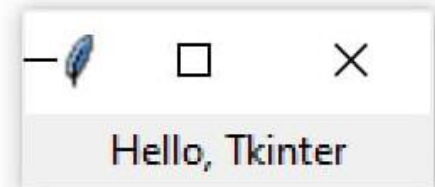
**If you used**  
from tkinter import \*  
Win=Tk()



# Adding a Widget

- Create a Label widget with the text "Hello, Tkinter" and assign it to a variable called greeting:

```
>>> greeting = tk.Label(text="Hello, Tkinter")  
>>> greeting.pack()
```



Each widget should end by using pack() method, to make the widget visible on window  
*Widget name*.pack()



# Control **Colors** in Label Widgets

- You can control Label text and background colors using the **foreground** and **background** parameters: fg,bg -bgcolor,fgcolor

```
>>> label = tk.Label(text="Hello, Tkinter",  
    foreground="white", # Set the text color to white  
    background="black" # Set the background color to black)
```

There are numerous valid color names, including: "red", "orange", "yellow", "green", "blue", and "purple"





# Control Colors in Label Widgets

- You can also specify a color using hexadecimal RGB values:

```
>>> label = tk.Label(text="Hello, Tkinter", background="#34A2FE")
```

- Shorthand **fg** and **bg** parameters to set the foreground and background colors:

```
>>> label = tk.Label(text="Hello, Tkinter", fg="white", bg="black")
```

Can use foreground (fg) & background (bg) with window

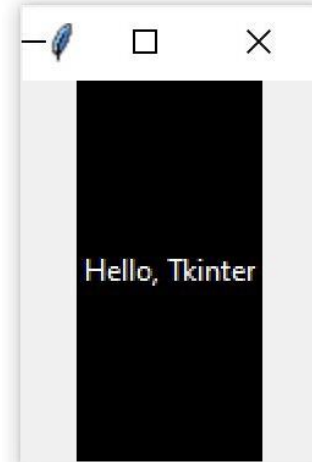
```
windowname['background']='yellow'  
windowname['bg']='black'  
w.bgcolor("pink")
```



# Control The **Width** and **Height** of a Label

- You can control the width and height of a label with the **width** and **height** parameters:

```
import tkinter as tk
window = tk.Tk()
window.title("Welcome")
window.geometry("200x100")
label = tk.Label(
    text="Hello, Tkinter",
    fg="white",
    bg="black",
    width=10,
    height=10)
label.pack()
window.mainloop()
```





# Displaying Clickable Buttons With Button Widgets

- **Button** widgets are used to display **clickable** buttons
- You can configure them to call a function whenever they are clicked
- **The similarities between Buttons and Labels:**
  - There are many similarities between Button and Label widgets. In many ways, a button is just a label that you can click!
  - The same keyword arguments that you use to create and style a Label will work with Button widgets





## Create and Style a Button

- Create a button with a blue background and yellow text. It also sets the width and height to 25 and 5 text units, respectively:

```
button = tk.Button(  
    text="Click me!",  
    width=25,  
    height=5,  
    bg="blue",  
    fg="yellow")  
button.pack()
```

Here is what the button looks like in a window:





# Using **command** option in a Button

- tkinter Button **command** option sets the **function or method** to be called when the **button is clicked**.
- To set a function for execution on button click, define a Python function, and assign this function name to the command option of Button:

```
import tkinter as tk
w=tk.Tk()
en1=tk.Entry(width=10,bg='pink')
en1.pack()
def fun1():
    nn=en1.get()
    lb3=tk.Label(text='hello ' +nn,width=20,height=4)
    lb3.place(x=200,y=100)
b2=tk.Button(text="display",width=10,height=5,command=fun1)
b2.place(x=100,y=20)
b1=tk.Button(text="Exit",width=10,height=5,command=lambda:w.destroy())
b1.place(x=20,y=20)
w.mainloop()
```

```
import tkinter as tk
w=tk.Tk()
def ex1():
    w.destroy()
b2=tk.Button(text="close",width=10,height=5,command=ex1)
b2.place(x=100,y=20)
w.mainloop()
```



# Getting User Input With Entry Widgets

- When to use Entry Widgets?

When you need to get a little bit of text from a user, like a name or an email address

- **Example:**

Create a widget with a blue background, some yellow text, and a width of 50 text units:

```
>>> entry = tk.Entry(fg="yellow", bg="blue", width=50)
        entry.pack()
```





# Operations on Entry Widgets

- There are three main operations that you can perform with Entry widgets:
  1. Retrieving text with `.get()`
  2. Deleting text with `.delete()`
  3. Inserting text with `.insert()`





# Operations on Entry Widgets

```
>>> import tkinter as tk
```

```
>>> window = tk.Tk()
```

- Now create a Label and an Entry widget:

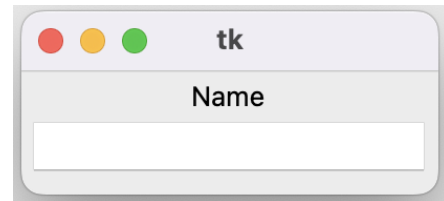
```
>>> label = tk.Label(text="Name")
```

```
>>> entry = tk.Entry()
```

- The Label describes what sort of text should go in the Entry widget. It does not enforce any sort of requirements on the Entry, but it tells the user what your program expects them to put there. You need to .pack() the widgets into the window so that they become visible:

```
>>> label.pack()
```

```
>>> entry.pack()
```







# Operations on Entry Widgets

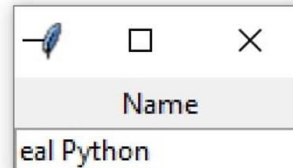
- You can use `.get()` to retrieve the text and assign it to a variable called `name` for example:

```
>>> name = entry.get()
>>> name
'Real Python'
```

Check example in  
slide 21

- You can delete text as well. This `.delete()` method takes an integer argument that tells Python which character to remove. For example, the code block below shows how `.delete(0)` deletes the first character from Entry:

```
>>> entry.delete(0)
```



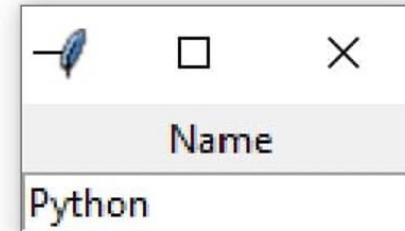


## Remove Several Characters From an Entry

- If you need to remove several characters from an Entry, then pass a second integer argument to `.delete()` indicating the index of the character where deletion should stop
- For example, the following code deletes **the first four letters** in Entry:

```
>>> entry.delete(0, 4)
```

The remaining text now reads Python:





# Remove Several Characters From an Entry

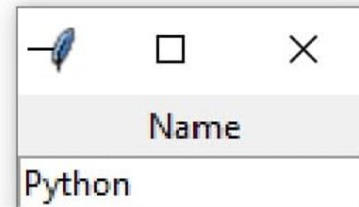
- Use the special constant `tk.END` for the second argument of `.delete()` to remove all text in Entry:

```
>>> entry.delete(0, tk.END)
```

- You will now see a blank text box:
- On the opposite end of the spectrum, you can also insert text into an Entry widget:

```
>>> entry.insert(0, "Python")
```

- The window now looks like this:



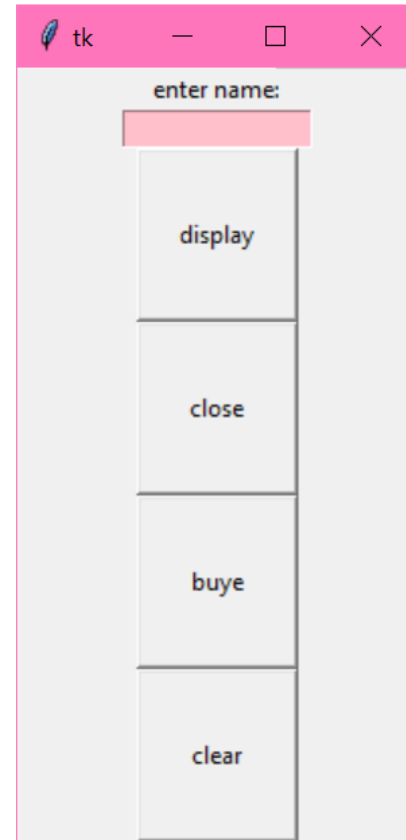


# Try and check

```

import tkinter as tk
w=tk.Tk()
lb1=tk.Label(text="enter name:")
lb1.pack()
en1=tk.Entry(width=15,bg='pink')
en1.pack()
def ins_t():
    en1.insert(0,"good bye")
def fun1():
    nn=en1.get()
    lb3=tk.Label(text='hello '+nn,width=15,height=4)
    lb3.pack()
def d_el():
    en1.delete(0,tk.END)
b2=tk.Button(text="display",width=10,height=5,command=fun1)
b2.pack()
def ex1():
    w.destroy()
b5=tk.Button(text='close',width=10,height=5,command=ex1)
b5.pack()
b6=tk.Button(text="buye",width=10,height=5,command=ins_t)
b6.pack()
b4=tk.Button(text="clear",width=10,height=5,command=d_el)
b4.pack()
w.mainloop()

```





# Getting Multiline User Input With Text Widgets

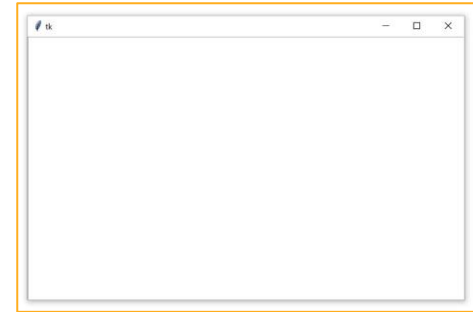
- The difference between Entry and Text widgets:
  - **Text** widgets are used for entering text, just like Entry widgets, the difference is that Text widgets may contain **multiple lines of text**
  - With a Text widget, a user can input a whole paragraph or even several pages of text





# Operations on Text Widgets

- Three main operations with Text widgets:
  1. Retrieve text with `.get()`
  2. Delete text with `.delete()`
  3. Insert text with `.insert()`



```
>>> import tkinter as tk
>>> window = tk.Tk()
>>> text_box = tk.Text()
>>> text_box.pack()
```

```
>>> from tkinter import*
>>> window = Tk()
>>> text_box = Text()
>>> text_box.pack()
```

- Text boxes are much larger than Entry widgets by default
- Here is what the window created above looks like:



# Retrieve the Text from a Text Widget

- Retrieve text from a Text widget is done by using `.get()`. However, calling `.get()` with no arguments does not return the full text in the text box like it does for Entry widgets. It raises an exception:

```
>>> text_box.get()
```

```
Traceback (most recent call last):
```

```
...
```

```
TypeError: get() missing 1 required positional argument: 'index1'
```

- `Text.get()` requires at least one argument. Calling `.get()` with a single index returns a single character. To retrieve several characters, you need to pass a **start index** and an **end index**. Indices in Text widgets work differently than in Entry widgets.





# Delete Characters from a Text Box

- `.delete()` is used to delete characters from a text box
- It works just like `.delete()` for Entry widgets
- There are two ways to use `.delete()`:
  1. With a **single argument**
  2. With **two arguments**



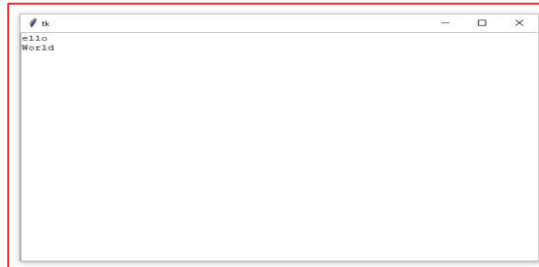


## Single-argument Version

- Using the single-argument version, you pass to `.delete()` the index of a single character to be deleted
- For example, the following deletes the first character, H, from the text box:

```
>>> text_box.delete("1.0") #1.0 mean line1 , first column(0)
```

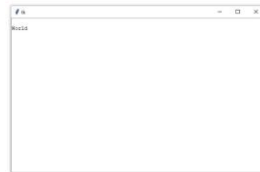
- The first line of text in the window now reads ello:





## Two-argument Version

- With the two-argument version, you pass two indices to delete a range of characters starting at the first index and up to, but not including, the second index
- For example:



Delete the remaining ello on the first line of the text box, use the indices "1.0" and "1.4":

```
>>> text_box.delete("1.0", "1.5") #can be without""  
>>> text_box.delete("1.0", END)  # DELETE ALL from first line
```

Notice that the text is gone from the first line. This leaves a blank line followed the word World on the second line:



# Insert Text into a text Box

- Insert text into a text box using `.insert()`:  

```
>>> text_box.insert("1.0", "Hello")
```
- This inserts the word Hello at the beginning of the text box, using the same "<line>.<column>" format used by `.get()` to specify the insertion position:



# Try and check

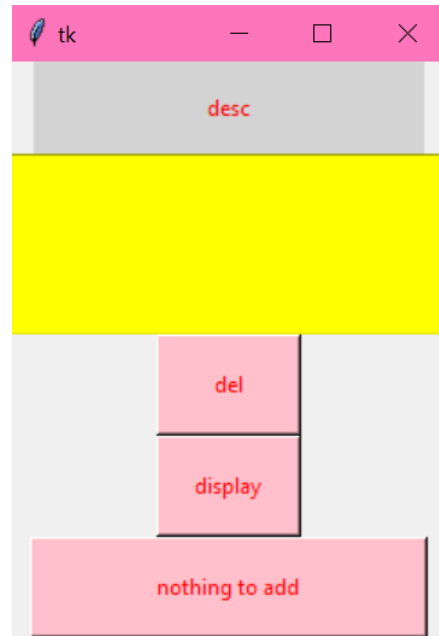
Insert(),get(),delete() with text

```

from tkinter import *
w=Tk()
lb1=Label(text="desc",width=30,height=3,fg="red",bg="light gray")
lb1.pack()
t1=Text(width=30,height=6,bg="yellow")
t1.pack()
def d_el():
    t1.delete(1.0,END)
def dis():
    x=t1.get(1.0,END)
    lb2=Label(text=x,fg="red",bg="light blue")
    lb2.pack()
def ins():
    t1.insert(1.0,"nothing")

b1=Button(text="del",width=10,height=3,fg="red",bg="pink",command=d_el)
b1.pack()
b2=Button(text="display",width=10,height=3,fg="red",bg="pink",command=dis)
b2.pack()
b3=Button(text="nothing to add",width=30,height=3,fg="red",bg="pink",command=ins)
b3.pack()
w.mainloop()

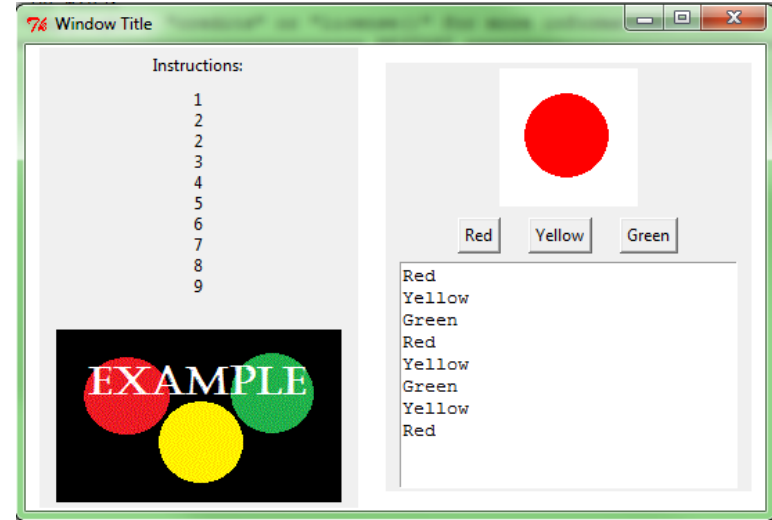
```





# Frame Widgets

- Frame widgets are important for organizing the **layout of your widgets** in an application





# Assigning Widgets to Frames With Frame Widgets

- Create a blank Frame widget and assigns it to the main application window:

```
>>> import tkinter as tk
```

```
>>> window = tk.Tk()
```

```
>>> frame = tk.Frame()
```

```
>>> #frame=tk.Frame(window)
```

```
>>> frame.pack()
```

```
>>> window.mainloop()
```







# The Main Use of Frames

- Frames are best thought of as **containers** for other widgets. You can assign a widget to a frame by setting the widget's master attribute:

```
>>> frame = tk.Frame()
>>> label = tk.Label(master=frame)
```
- Frame widgets are great for organizing other widgets in a logical manner
- Related widgets can be assigned to the same frame so that, if the frame is ever moved in the window, then the related widgets stay together

- All four of the widget types—Label, Button, Entry, and Text—have a **master** attribute that's set when you instantiate them. That way, you can control which Frame a widget is assigned to

**master** –refer to target frame which widget displayed in, better to used when we have more than one frame



## Try and check:

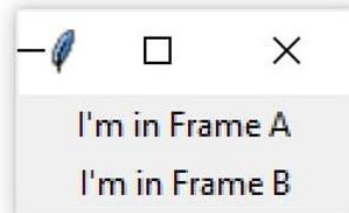
- Write a script that creates two Frame widgets called `frame_a` and `frame_b`. In this script, `frame_a` contains a label with the text *"I'm in Frame A"*, and `frame_b` contains the label *"I'm in Frame B"*





## Script:

```
>>> import tkinter as tk
>>> window = tk.Tk()
>>> frame_a = tk.Frame(window)
>>> frame_b = tk.Frame(window)
```



```
>>> label_a = tk.Label(master=frame_a, text="I'm in Frame A")
>>> label_a.pack()
```

```
>>> label_b = tk.Label(master=frame_b, text="I'm in Frame B")
>>> label_b.pack()
>>> frame_a.pack()
>>> frame_b.pack()
>>> window.mainloop()
```

**Question:** What happens when you swap the order of `frame_a.pack()` and `frame_b.pack()`?



# Controlling Layout With Geometry Managers

- Application layout in Tkinter is controlled with **geometry managers**
- Examples of geometry manager in Tkinter:
  - `.pack()`
  - `.place()`
  - `.grid()`
- Each window or Frame in your application can use only one geometry manager
- However, different frames can use different geometry managers, even
  - if they are assigned to a frame or window using another geometry manager

- (in default widgets geometry will be in center)
- Can used for other widgets not just frame
- `.pack()` , `.place()` , `.gride()` not used together for widgest in window



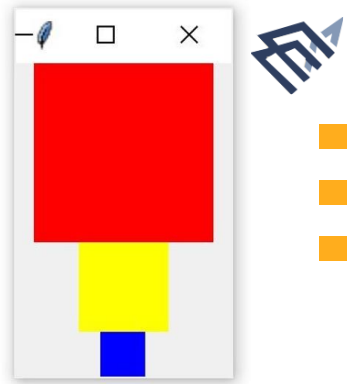
# The .pack() Geometry Manager

- The **packing algorithm** to place widgets in a Frame or window in a specified order
- The packing algorithm has two primary steps:
  1. Compute a rectangular area called a **parcel** that is just tall (or wide) enough to hold the widget and fills the remaining width (or height) in the window with blank space
  2. Center the widget in the parcel unless a different location is specified



## Ex: .pack() three Label widgets into a Frame

```
>>> import tkinter as tk
>>> window = tk.Tk()
>>> frame1 = tk.Frame(master=window, width=100, height=100, bg="red")
>>> frame1.pack()
>>> frame2 = tk.Frame(master=window, width=50, height=50, bg="yellow")
>>> frame2.pack()
>>> frame3 = tk.Frame(master=window, width=25, height=25, bg="blue")
>>> frame3.pack()
>>> window.mainloop()
```





# Fill Horizontal and Vertical Direction

- **Fill** keyword argument is used to specify in which **direction** the frames should fill
- The options are:
  1. **tk.X** to fill in the horizontal direction
  2. **tk.Y** to fill vertically
  3. **tk.BOTH** to fill in both directions
- How to stack three frames so that each one fills the whole window horizontally?

*Widget name.pack(fill=X)*

# Script:

```
>>> import tkinter as tk
>>> window = tk.Tk()
>>> frame1 = tk.Frame(master=window, height=100, bg="red")
>>> frame1.pack(fill=tk.X)
>>> frame2 = tk.Frame(master=window, height=50, bg="yellow")
>>> frame2.pack(fill=tk.X)
>>> frame3 = tk.Frame(master=window, height=25, bg="blue")
>>> frame3.pack(fill=tk.X)
>>> window.mainloop()
```







## Side Argument Options

- The **side** keyword argument of `.pack()` specifies on which side of the window the widget should be placed. These are the available options:
  - `tk.TOP`
  - `tk.BOTTOM`
  - `tk.LEFT`
  - `tk.RIGHT`

- *Widget name.pack(side=LEFT , fill=Y)*

- The default option is CENTER if not used side argument



# The .place() Geometry Manager

- Use `.place()` to control the **precise location** that a widget should occupy in a window or Frame
- Must provide two keyword arguments, `x` and `y`, which specify the x- and y-coordinates for the top-left corner of the widget
- Both `x` and `y` are measured in pixels, not text units
- The **origin**, where `x` and `y` are both 0, is the top-left corner of the Frame or window *Widget name.place(x= , y= )*

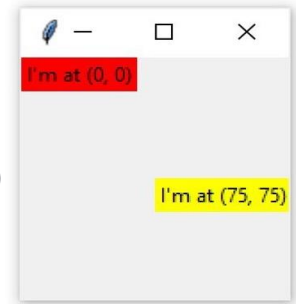
- You can think of the `y` argument of `.place()` as the number of pixels from the top of the window, and the `x` argument as the number of pixels from the left edge of the window





## Example of using .place() geometry manager:

```
>>> import tkinter as tk
>>> window = tk.Tk()
>>> frame = tk.Frame(master=window, width=150, height=150)
>>> frame.pack()
>>> label1 = tk.Label(master=frame, text="I'm at (0, 0)", bg="red")
>>> label1.place(x=0, y=0)
>>> label2 = tk.Label(master=frame, text="I'm at (75, 75)", bg="yellow")
>>> label2.place(x=75, y=75)
>>> window.mainloop()
```





## Drawback of .place()

1. Layout can be difficult to manage with .place() especially if the application has many widgets
2. Layouts created with .place() are not responsive (they do not change as the window is resized)

.pack() is usually a better choice than .place(), but even .pack() has some downsides. The placement of widgets depends on the order in which .pack() is called, so it can be difficult to modify existing applications without fully understanding the code controlling the layout. The .grid() geometry manager solves a lot of these issues, as you'll see in the next section.





# The .grid() Geometry Manager

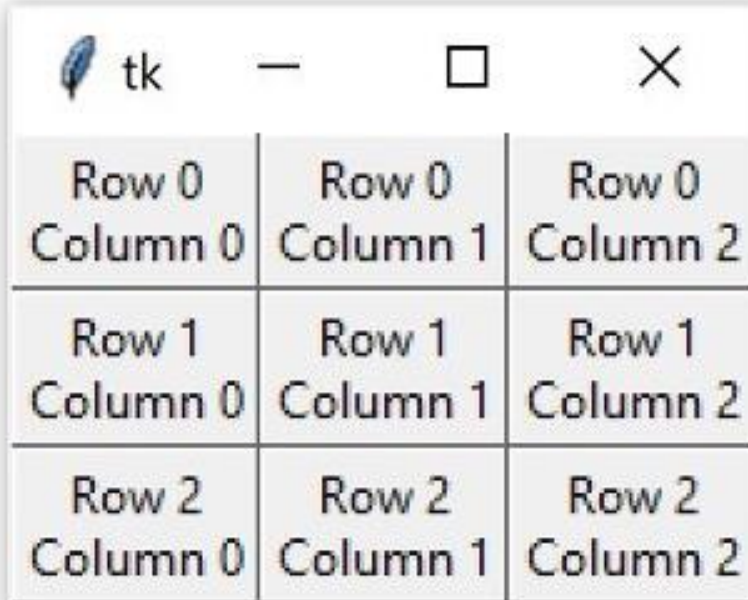
- The most used geometry manager is .grid(), which provides all the power of .pack() in a format that is easier to understand and maintain
- It works by splitting a window or Frame into rows and columns
- You specify the location of a widget by calling .grid() and passing the row and column indices to the **row** and **column** keyword arguments, respectively

Both row and column indices start at 0, so a row index of 1 and a column index of 2 tells .grid() to place a widget in the third column of the second row.



## Example of Using .grid()

- Creates a  $3 \times 3$  grid of frames with Label widgets packed into them:



Row 0 Column 0	Row 0 Column 1	Row 0 Column 2
Row 1 Column 0	Row 1 Column 1	Row 1 Column 2
Row 2 Column 0	Row 2 Column 1	Row 2 Column 2

*Widget name*.grid(row= , column= )



# Example of Using .grid()

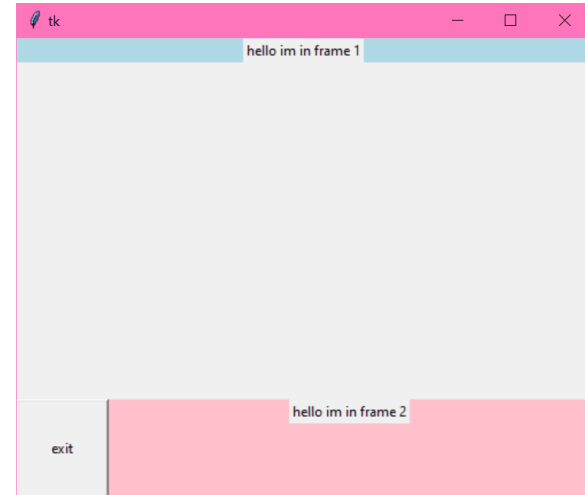
```
from tkinter import *
root = Tk()
frame = Frame(root)
redbutton = Button(frame, text = 'Red', fg='red')
#redbutton.pack( side = LEFT)
redbutton.grid(row=0,column=0)
greenbutton = Button(frame, text = 'Brown', fg='brown')
#greenbutton.pack( side = LEFT )
greenbutton.grid(row=0,column=1)
bluebutton = Button(frame, text = 'Blue', fg='blue')
#bluebutton.pack( side = LEFT )
bluebutton.grid(row=0,column=2)
blackbutton = Button(frame, text = 'Black', fg='black')
#blackbutton.pack( side = LEFT)
blackbutton.grid(row=0,column=3)
frame.pack()
root.mainloop()
```



```
from tkinter import *
```

```
root = Tk()
root.geometry("500x400")
f1=Frame(root,width=200,height=100,bg="light blue")
f2=Frame(root,width=70,height=30,bg="pink")
b1=Label(master=f1,text="hello im in frame 1")
b1.pack()
bt1=Button(master=f2,text="exit",width=10,height=5,command=lambda:
root.destroy())
bt1.pack(side=LEFT)
b2=Label(master=f2,text="hello im in frame 2")
b2.pack()

f2.pack(side=TOP,fill=X)
f1.pack(side=TOP,fill=X)
root.mainloop()
```







# Exercises



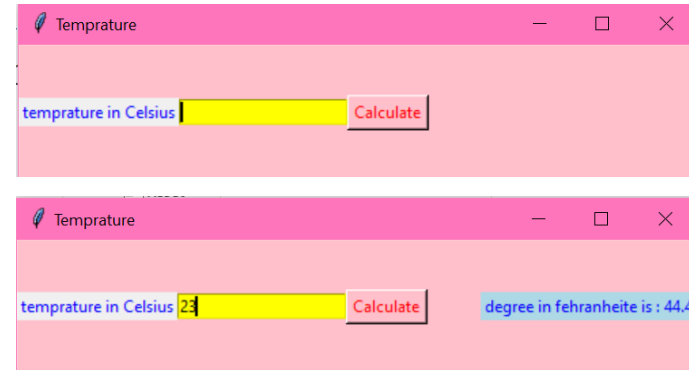
# Exercise 1: Temperature Converter Application

- Write a python program to build a temperature converter application that allows the user to input temperature from Celsius to Fahrenheit and click a button to convert that temperature to degrees Celsius that shown in a label , using methods and attributes to be as a figure.

Note:

$$F = (C * 9/5) + 32$$

Window geometry 600x100



output

```
from tkinter import*
```

```
root=Tk()
```

```
root.title("Temprature")
```

```
root.geometry("600x100")
```

```
root['bg']="pink"
```

```
l1=Label(text="temprature in Celsius",fg="blue")
```

```
l1.pack(side=LEFT)
```

```
t1=Entry(bg="yellow")
```

```
t1.pack(side=LEFT)
```

```
def c_to_f():
```

```
    C=eval(t1.get())
```

```
    F=(C*9/5)+3
```

```
    L1=Label(text="degree in fehranheite is : "+ str(F),fg="blue",bg="light blue")
```

```
    L1.pack(side=RIGHT)
```

```
b=Button(text="Calculate",bg="pink",fg="red",command=c_to_f)
```

```
b.pack(side=LEFT)
```

```
root.mainloop()
```



## Exercise 2: summation two numbers

Write a python program to calculate and display **summation of** two numbers given by a user , the result should display on label when a user click on button (calculate), delete all entries when click button (clear) ,close a window when click button(Exit) ,no resizable for a window,the window geometry 300x200, use grid() geometry for widget , and other attributes and methods as shown in a figure1,the output should be as figure2

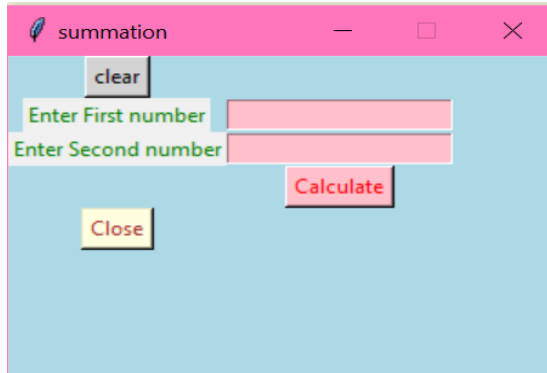


figure1

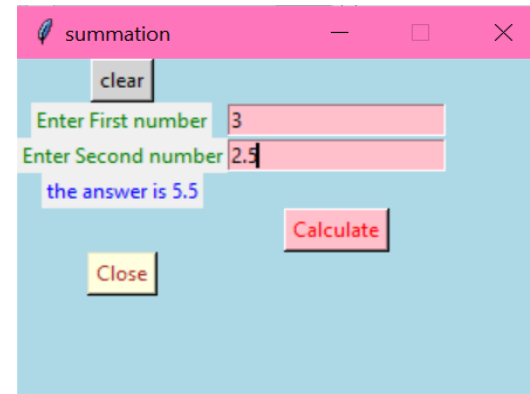


figure2

```

from tkinter import*
root=Tk()
root.title("summation")
root.geometry("300x200")
root.resizable(False,False)
root['bg']="lightblue" #use this code to change the background color
l1=Label(text="Enter First number",fg="Green")
#l1.place()
l1.grid(row=1,column=0)
t1=Entry(bg="pink")
t1.grid(row=1,column=1)
l2=Label(text="Enter Second number",fg="Green")
l2.grid(row=2,column=0)
t2=Entry(bg="pink")
t2.grid(row=2,column=1)
def sum():# function for summation to numbers given by user
    num1=eval(t1.get())# get function change the number to string
    num2=eval(t2.get())
    s=num1+num2
    L1=Label(text="the answer is " + str(s),fg="blue")
    L1.grid(row=3,column=0)
def clear():#function to clear entries
    t1.delete(0,END)
    t2.delete(0, END)
def exit():#function to close window
    root.destroy()
b=Button(text="Calculate",bg="pink",fg="red",command=sum) #use command to use the function
b.grid(row=4,column=1)
b2=Button(text="clear",bg="light gray",fg="black",command=clear) #use command to use the function
b2.grid(row=0,column=0)
b3=Button(text="Close",bg="light yellow",fg="brown",command=exit) #use command to use the function
b3.grid(row=5,column=0)
root.mainloop()

```



## Exercise 3: calculate Age

Write a python program to calculate and display **Age** after read **current year**, **Birth year** from a user , the result for age should display on label when a user click on button (Age), **use default attributes and methods for window and widgets**, (all widgets should be in a frame, background color for frame=light yellow , should be fill Both x,y )

Window geometry 200x150

Expected output for different entries

The screenshot shows a Tkinter window titled "Age" with a light yellow background. It contains two input fields: "enter current year" and "enter Birth year", each with a corresponding text entry box. Below these fields is a red button labeled "Age".

```
from tkinter import *
win=Tk()
win.title("Age")
win.geometry("200x150")
fr=Frame(master=win,bg="light yellow")
lb1=Label(master=fr,text="enter current year")
lb1.pack()
e1=Entry(master=fr)
e1.pack()
lb2=Label(master=fr,text="enter Birth year")
lb2.pack()
e2=Entry(master=fr)
e2.pack()
def age():
    cy=int(e1.get())
    by=int(e2.get())
    ag=cy-by
    label_age=Label(master=fr,text=ag)
    label_age.pack()
b1=Button(master=fr,text="Age",width=10,height=3,fg="red",bg="pink",command=age)
b1.pack()
fr.pack(fill=BOTH)
win.mainloop()
```

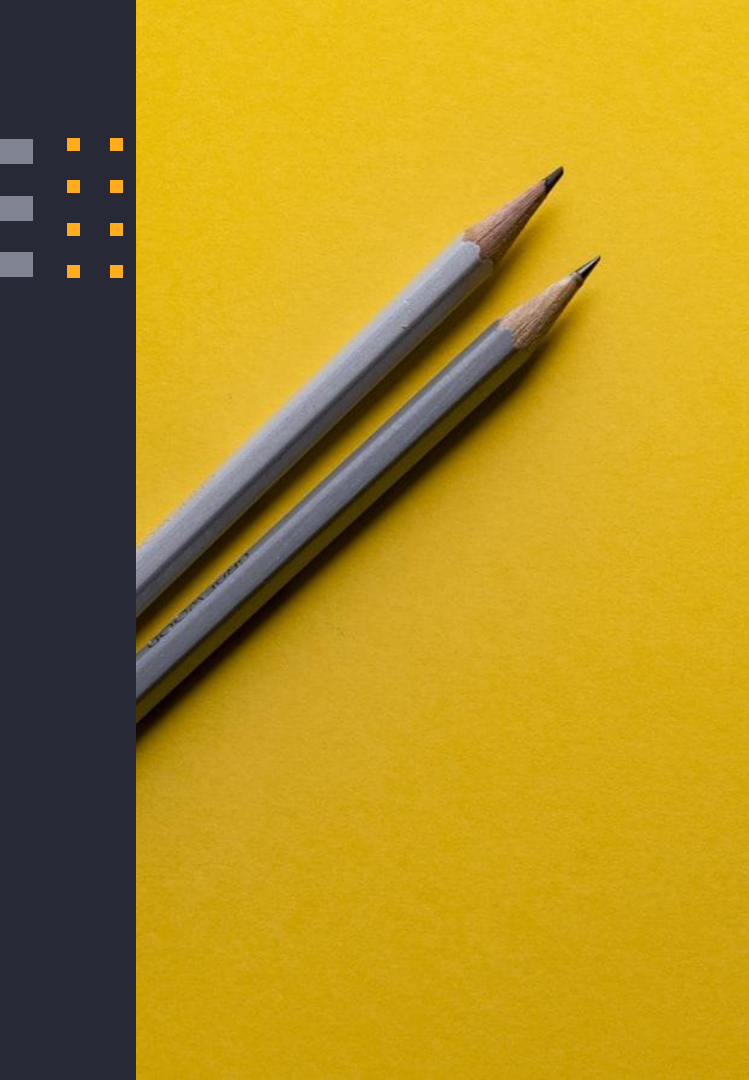


# References

- Chapter “Graphical User Interfaces” of Python Basics: A Practical Introduction to Python 3
- Shipman, J. W. (2013). Tkinter 8.5 reference: a GUI for Python. *New Mexico Tech Computer Center*, 54.
- <https://realpython.com/python-gui-tkinter/>







# Thanks!

**Any questions?**