# Creating Graphical User Interfaces (GUI)

DIT102 Programming II

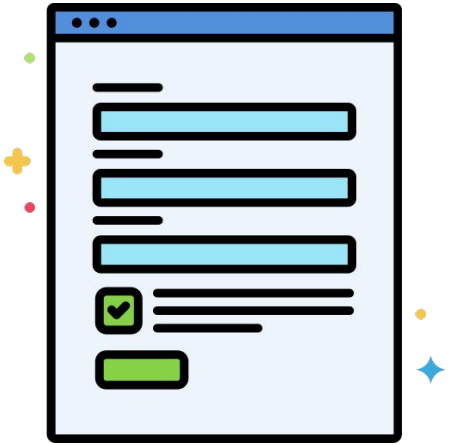
Asst. Prof. Dr. Parkpoom Chaisiriprasert

# GUI programming in Python

Tkinter is pronounced as tea-kay-inter and serves as the Python interface to Tk, the GUI toolkit for Tcl/Tk.

Python implements Tkinter as a module, serving as a wrapper for C extensions that utilize Tcl/Tk libraries.

Tkinter allows you to develop desktop applications, making it a valuable tool for GUI programming in Python



# Creating a window

First, import the tkinter module as tk to the program:

```
import tkinter
```

Second, create an instance of the tk.Tk class that will create the application window:

```
root = tkinter .Tk()
```

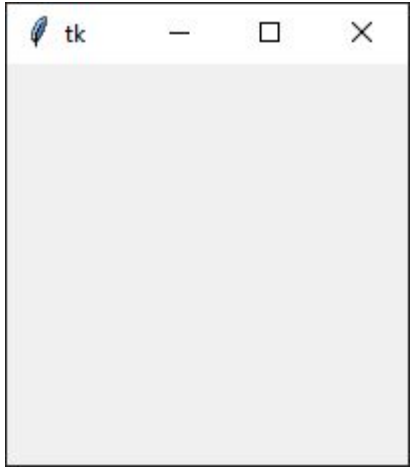
Third, call the mainloop() method of the main window object

```
root.mainloop()
```

# Creating a window

The following program shows how to display a window on the screen. The root window has a title that defaults to tk. It also has three system buttons including Minimize, Maximize, and Close.

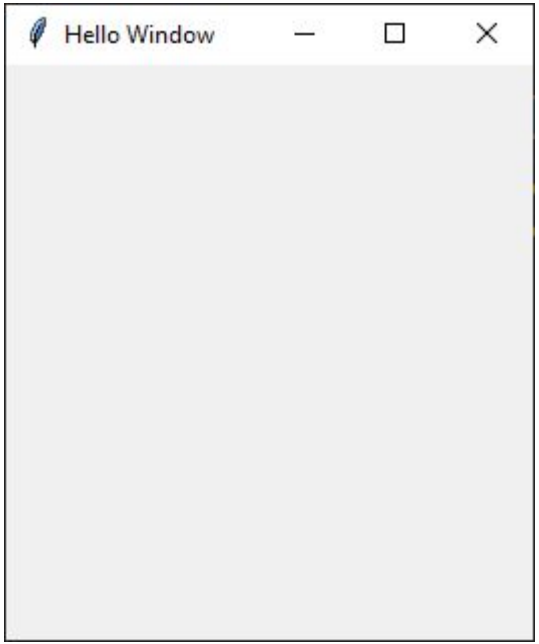
```
import tkinter

root = tkinter .Tk()

root.mainloop()
```



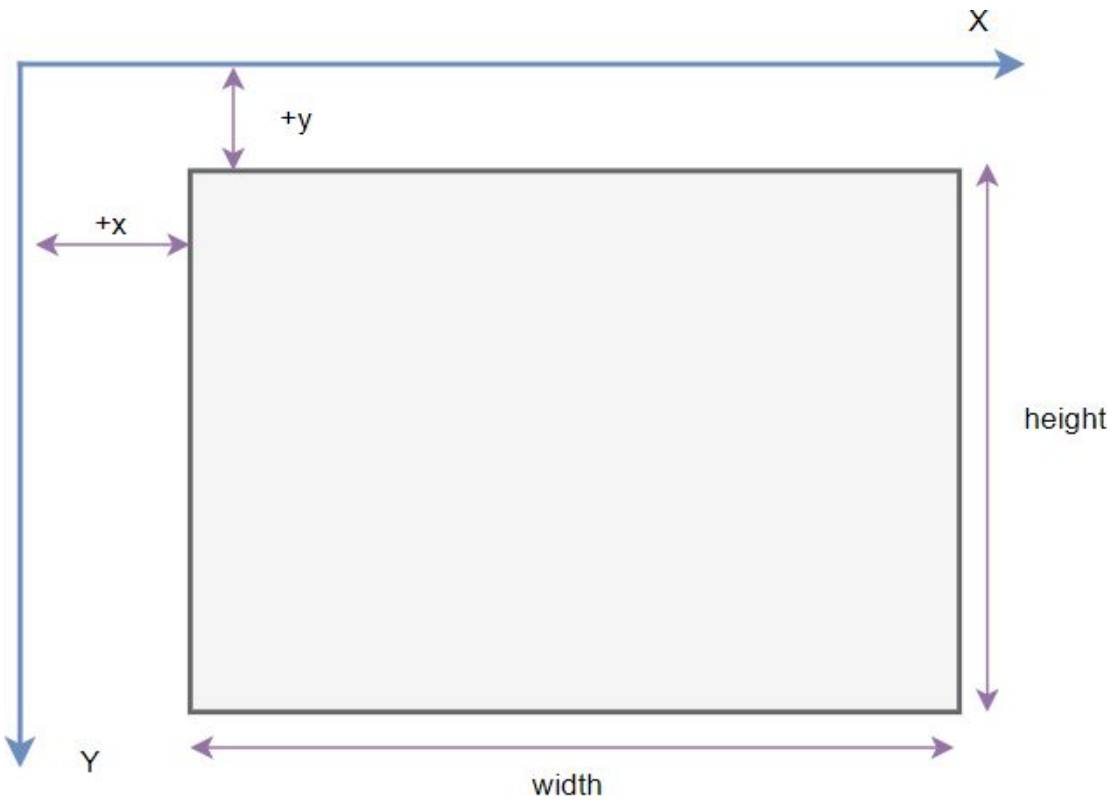
# Changing the window title

To change the window's title, you use the title() method like this:

```
import tkinter
root = tkinter .Tk()
root.title('Hello Window')
root.mainloop()
```



# Changing window size and location




```
window.geometry('widthxheight±x±y')
```

# Changing window size and location

The following example changes the size of the window to 600x400 and the position of the window to 50 pixels from the top and left of the screen:

```
import tkinter
root = tkinter .Tk()
root.title('Hello Window')
root.geometry('600x400+50+50')
root.mainloop()
```

# Changing window size and location

Sometimes, you may want to full the window on the screen.

```
import tkinter as tk
root = tk.Tk()
root.title('Hello Window')

window_width = root.winfo_screenwidth()
window_height = root.winfo_screenheight()
root.geometry(f'{window_width}x{window_height}+{0}+{0}')

root.mainloop()
```

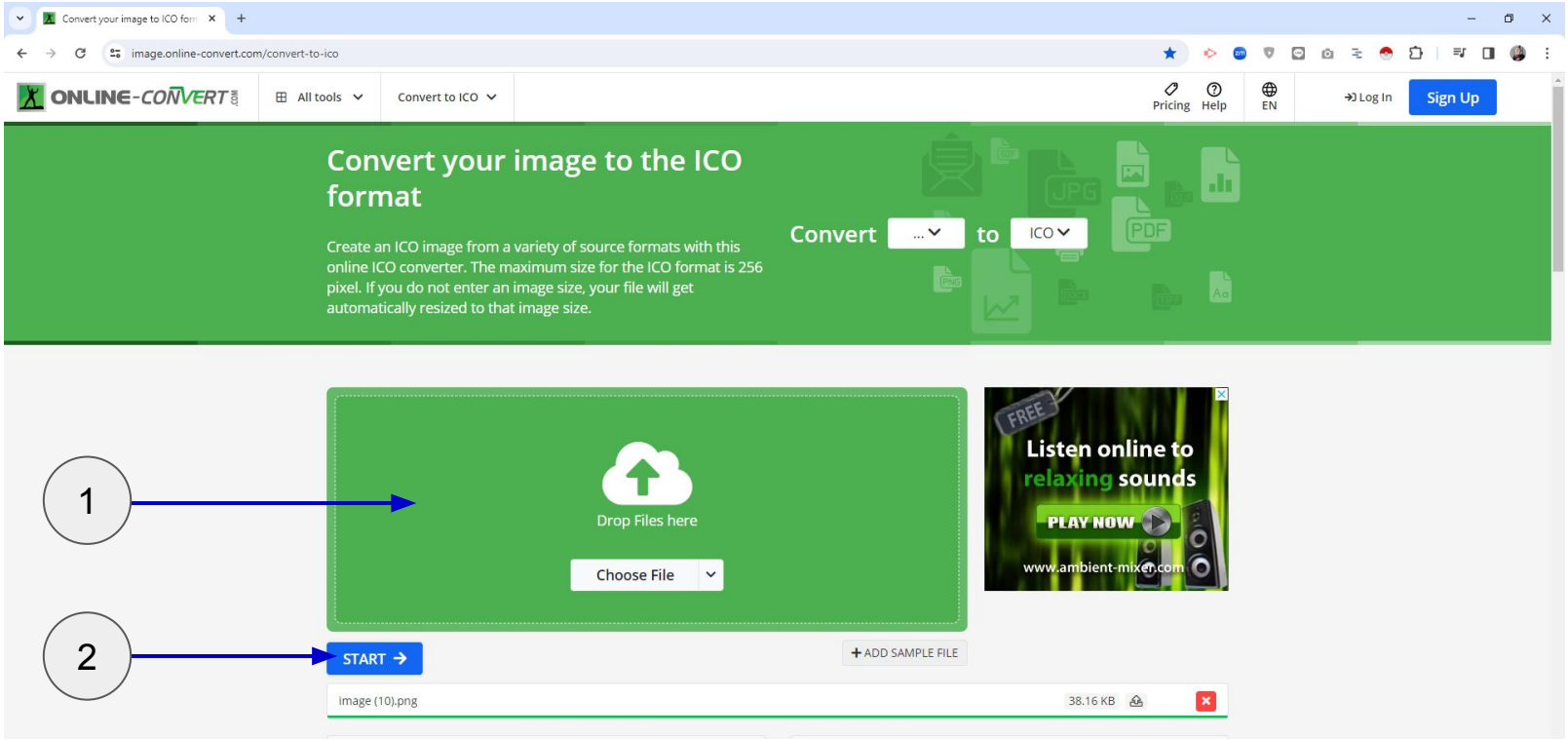
# Changing the default icon

- Sample image: https://drive.google.com/drive/folders/1YtFQwRZlRyUCSk5MZBRQf1q3esicaayt?usp=sharing
- Prepare an image in the .ico format. If you have the image in other formats like png or jpg, you can convert it to the .ico format. There are many online tools that allow you to do it quite easily. https://image.online-convert.com/convert-to-ico

- Place the icon in a folder that can be accessible from the program.

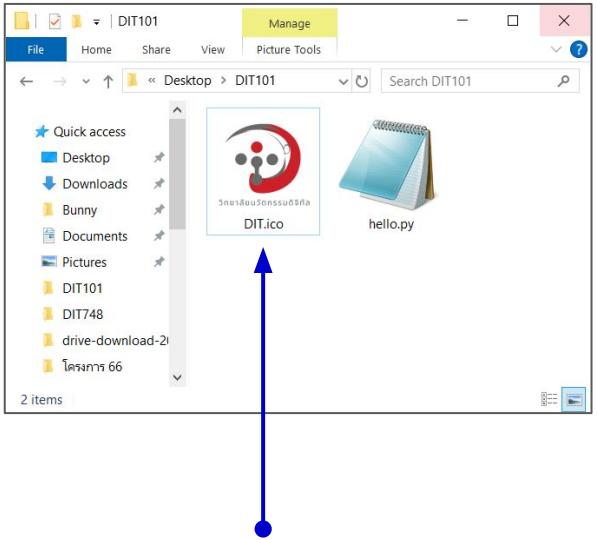
# Prepare an image in the .ico format

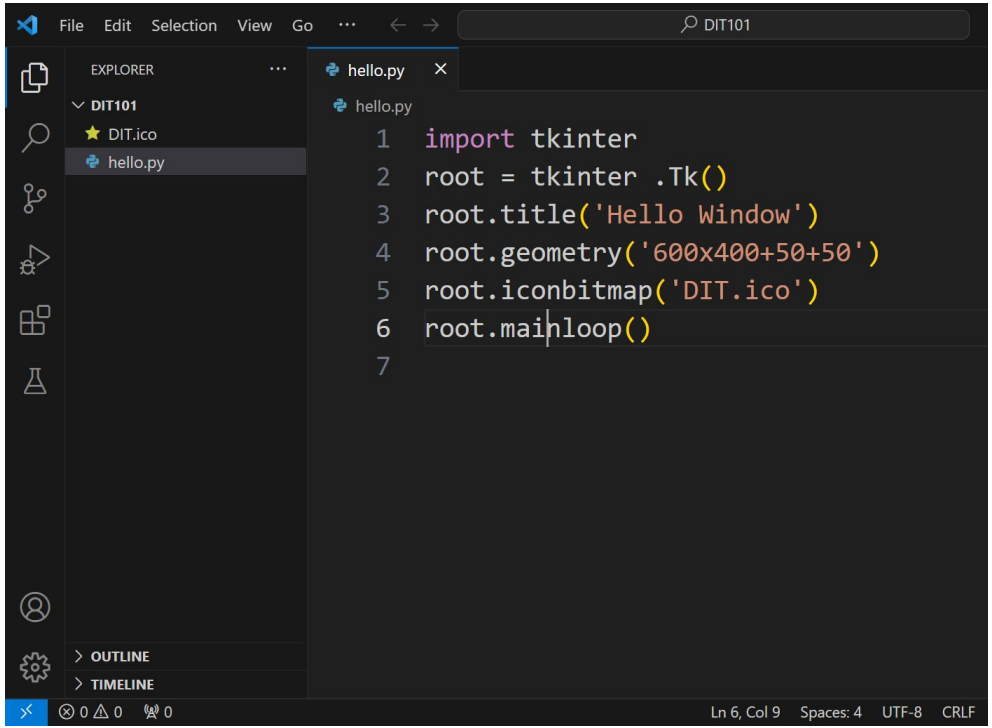
Choose your file, start convert and download



# Place the icon in a folder

Place the icon in a folder that can be accessible from the program.

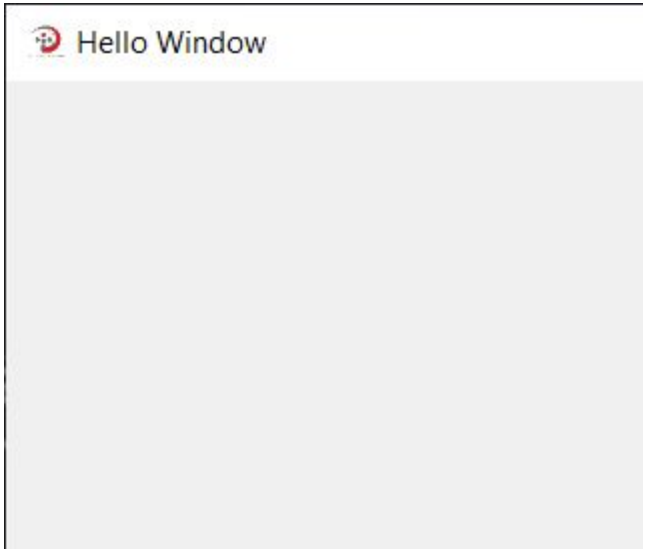


# Changing the default icon

Call the iconbitmap() method of the window object.

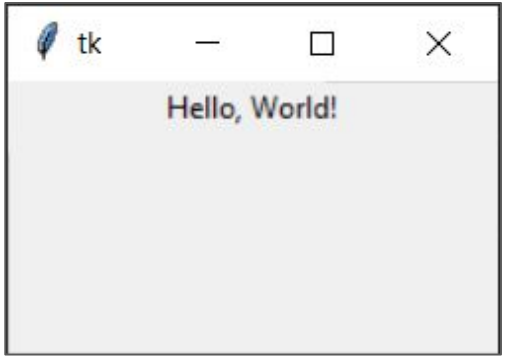
```
import tkinter
root = tkinter .Tk()
root.title('Hello Window')
root.geometry('600x400+50+50')
root.iconbitmap('DIT.ico')
root.mainloop()
```



# Displaying a label

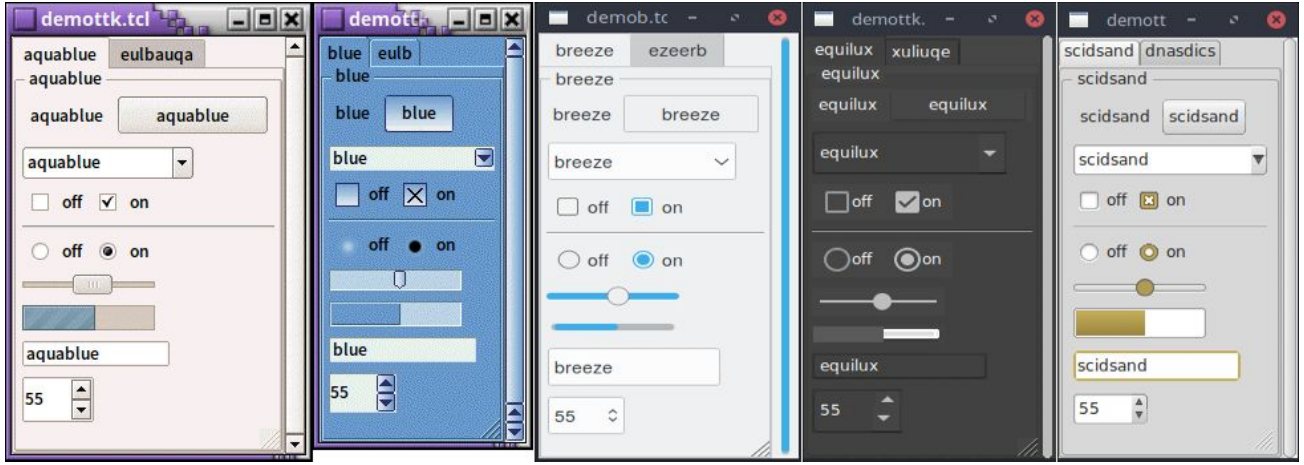
Now, let's place a component on the window. In Tkinter, these components are referred to as **widgets**.

```
import tkinter as tk

root = tk.Tk()

message = tk.Label(root, text="Hello, World!")

message.pack()

root.mainloop()
```



# Ttk (Themed Tkinter) widgets

Ttk widgets are an extension of the Tkinter library in Python. Tkinter is the standard GUI (Graphical User Interface) toolkit that comes with Python. Ttk widgets provide more modern and visually appealing user interface components compared to the standard Tkinter widgets.



# Tk and Ttk widgets

The tkinter.ttk module contains all the new ttk widgets. It's a good practice to always use themed widgets whenever they're available.

```
import tkinter as tk

from tkinter import ttk
```

The difference is that "from <module/file> import <class/module>" is used for importing some specific thing from that file/module. In the other hand "Import<module> is used for importing the whole module/file.

# Tk and Ttk widgets

They look similar. However, their appearances depend on the platform on which the program runs.

```
import tkinter as tk
from tkinter import ttk
root = tk.Tk()
tk.Label(root, text='Classic Label').pack()
ttk.Label(root, text='Themed Label').pack()
root.mainloop()
```

# Tk and Ttk widgets

## Tk and Ttk widgets

- Label
- Entry
- Button
- Checkbutton
- Frame
- LabelFrame
- Menubutton
- PanedWindow
- Radiobutton
- Scale
- Scrollbar
- Spinbox

## Ttk widgets

- Combobox
- Notebook
- Progressbar
- Separator
- Sizegrip
- Treeview

# Set Options for a Ttk Widget

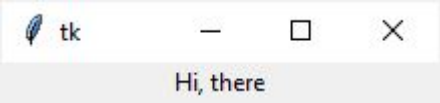
Ttk widget using the widget constructor, a dictionary index, and config() method. When working with themed widgets, you often need to set their attributes such as text and image.

- Use the **widget constructor** during the widget's creation.
- Set a property value using a **dictionary index** after creating the widget.
- Call the **config()** method with keyword arguments.

# The widget constructor when creating the widget

The following illustrates how to use the widget constructor to set the text option for the Label widget:

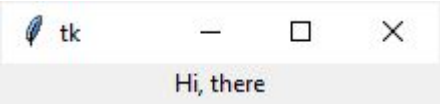
```
import tkinter as tk
from tkinter import ttk

root = tk.Tk()
ttk.Label(root, text='Hi, there').pack()
root.mainloop()
```



# A dictionary index after widget creation

The following program shows the same label but uses a dictionary index to set the text option for the Label widget:

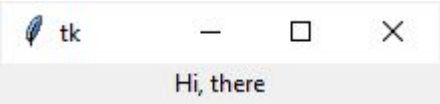
```
import tkinter as tk
from tkinter import ttk

root = tk.Tk()
label = ttk.Label(root)
label['text'] = 'Hi, there'
label.pack() # place a label on the root window
root.mainloop()
```



# The config() method with keyword arguments

The following program illustrates how to use the config() method with a keyword argument to set the text option for the label:

```
import tkinter as tk
from tkinter import ttk

root = tk.Tk()
label = ttk.Label(root)
label.config(text='Hi, there')
label.pack()
root.mainloop()
```



# Label widget

It is used to display a text or image on the screen. To use a Label widget, you use the following general syntax.

The main window in Tkinter is called root. But you can use any other name like Master.

```
master= tk.Tk()
label = ttk.Label(master, **options)
```

The Label widget has many options that allow you to customize its appearance

```
label = ttk.Label(master, text='Hi, there')
```

# Setting a specific font for the Label

To set a particular font for a label, you pass the font keyword argument to the Label constructor like this:

font name

```
font = ('font name', font_size)
```

font size

```
label = ttk.Label(master, text='Hi, there', font=("Helvetica", 14))
```

# Displaying an image

First, create a PhotoImage widget by passing the path of the photo to the PhotoImage constructor.

Variables used to store image data

The path to the image location stored on the computer.

```
variable = tk.PhotoImage(file='./path/filename')
```

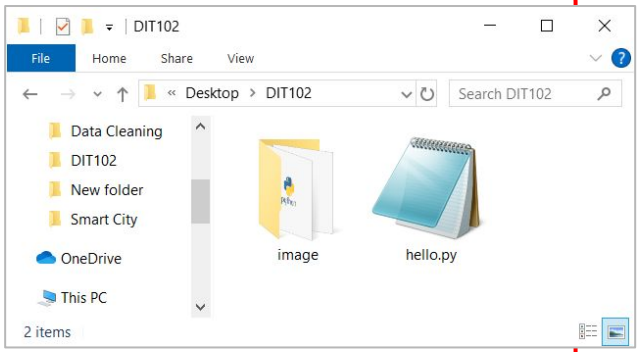
Image file name

```
photo = tk.PhotoImage(file='./image/python.png')
```

# Displaying an image

Assign the PhotoImage to the image option of the Label widget.

```
import tkinter as tk
from tkinter import ttk
root = tk.Tk()
root.geometry('300x200')
root.title('Label Widget Image')

photo = tk.PhotoImage(file='./image/python.png')

label = ttk.Label(root, image=photo)
label.pack()

root.mainloop()
```




# Exercise

Write a program to create a screen as shown in the example screenshot that follows.

# Entry widget

The Entry widget allows you to enter a single-line text. In Tkinter, to create a textbox, you use the Entry widget.

The master is the parent widget on which you place the button.

The options is one or more keyword arguments used to configure the Entry widget.

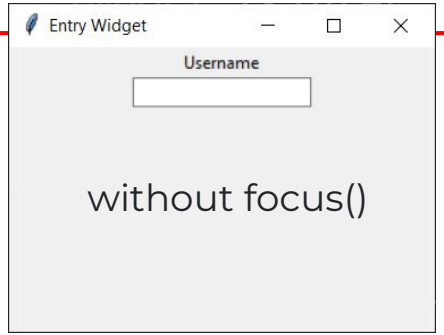
```
master = tk.Tk()
textbox = ttk.Entry(master, **options)
```

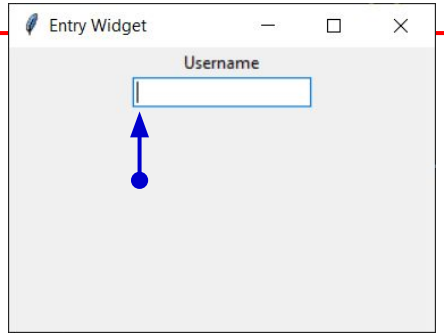
Typically, you associate the current value of the textbox with a StringVar object.

```
textbox = ttk.Entry(master, textvariable = tk.StringVar())
```

# The focus to the first Entry widget

```
import tkinter as tk
from tkinter import ttk
root = tk.Tk()
root.geometry('300x200')
root.title('Entry Widget')
ttk.Label(root, text='Username').pack()
textbox = ttk.Entry(root)
textbox .focus()
textbox .pack()
root.mainloop()
```



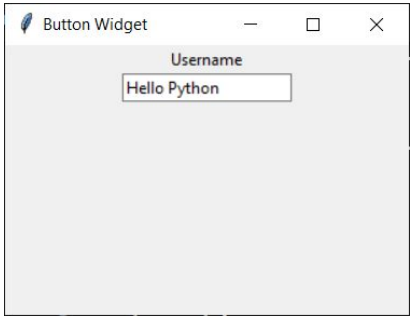




To provide a better user experience, you can move the focus to the first Entry widget after the window appears. Once the Entry widget has focus, it's ready to accept the user input.

# Set default text of a Entry Widget

To insert some text into an Entry field in Tkinter, call the insert() method and specify the index and value as arguments.

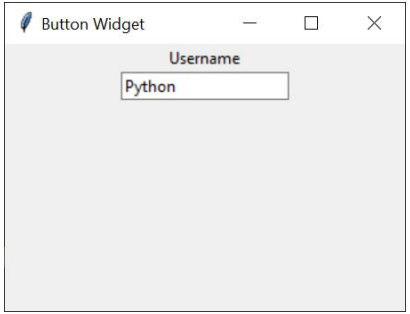
```
import tkinter as tk
from tkinter import ttk
root = tk.Tk()
root.geometry('300x200')
root.title('Button Widget')
tk.Label(root, text='Username').pack()
textbox = ttk.Entry(root)
textbox.insert(0,'Hello Python')
textbox .pack()
root.mainloop()
```

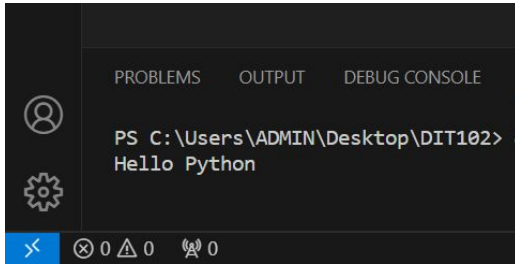


# Get the current text of a Entry Widget

To get the current value of the entry widget, use the get() method.

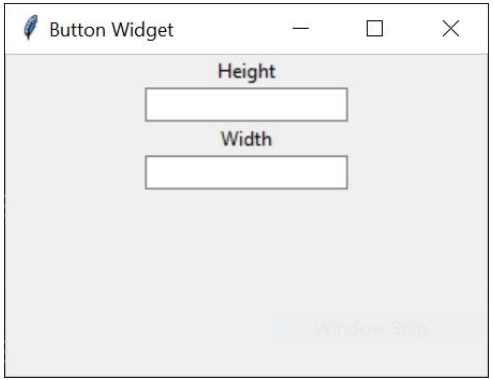
```
import tkinter as tk
from tkinter import ttk
root = tk.Tk()
root.geometry('300x200')
root.title('Button Widget')
tk.Label(root, text='Username').pack()
textbox = ttk.Entry(root)
textbox.insert(0,'Python')
text = textbox .get()
print('Hello', text)
textbox .pack()
root.mainloop()
```



# Exercise

Write a program to create a screen as shown in the example screenshot that follows.



# Button widget

Button widgets represent a clickable item in the applications. Typically, you use text or an image to display the action that will be performed when clicked.

The master is the parent widget on which you place the button.

The text is the label of the button.

```
master= tk.Tk()
button = ttk.Button(master, text, command)
```

The command specifies a callback function that will be called automatically when the button is clicked.

# Button Example


```
import tkinter as tk
from tkinter import ttk
root = tk.Tk()
root.geometry('300x200')
root.title('Button Widget')

button =  ttk.Button(root,text="Button")
button.pack()
root.mainloop()
```

# Button widget

In the normal state, the button will respond to mouse events and keyboard presses. The button can also have the disabled state, meaning that button is grayed out and doesn't respond to mouse events or keyboard presses.

```
# set the disabled flag
button.state(['disabled'])
```

```
# remove the disabled flag
button.state(['!disabled'])
```

# Tkinter image button

Assign the PhotoImage to the image option of the Button widget.

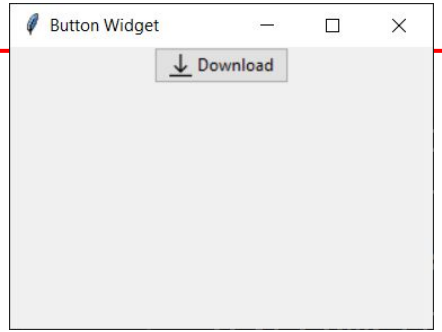
```
photo = tk.PhotoImage(file='./image/download.png')
```


```
import tkinter as tk
from tkinter import ttk
root = tk.Tk()
root.geometry('300x200')
root.title('Button Widget')
photo = tk.PhotoImage(file='./image/download.png')
button =  ttk.Button(root,image=photo )
button.pack()
root.mainloop()
```

# Tkinter image and text button

Assign the PhotoImage to the image option of the Button widget.

```
photo = tk.PhotoImage(file='./image/download.png')
```




```
import tkinter as tk
from tkinter import ttk
root = tk.Tk()
root.geometry('300x200')
root.title('Button Widget')
photo = tk.PhotoImage(file='./image/download.png')
button =  ttk.Button(root,image=photo,text='Download',compound=tk.LEFT)
button.pack()
root.mainloop()
```

Use the compound option if you want to display both text and image on a button.

# Button Response

The command option associates the button's action with a function or a method of a class. When click or press the button.

It can use a lambda expression for one expression.

```
button =  ttk.Button(root,text="Button",command=lambda:expression)
```

It'll automatically invoke a callback function.

```
def callback():
    # do something

button = ttk.Button(root,text="Button",command=callback)
```

It is called automatically when the button is clicked.

# Button Example


```
import tkinter as tk
from tkinter import ttk
root = tk.Tk()
root.geometry('300x200')
root.title(Button Widget)

button =  ttk.Button(root,text="Exit",command=lambda: root.quit())
button.pack()
root.mainloop()
```

The command of the button is assigned to a lambda expression that closes the root window.

# Tkinter Command Binding

To make the application more interactive, the widgets need to respond to the events such as Mouse clicks and Key presses that associates a callback with an event of a widget.

To use the command binding, you follow these steps:

1. define a function as a callback.
2. assign the name of the function to the command option of the widget.

# Define and Assign a function as a callback

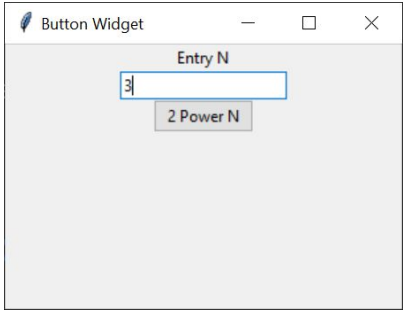
The following program illustrates how to associate the button_clicked() callback function with the Button widget:

```
import tkinter as tk
from tkinter import ttk
root = tk.Tk()

def button_clicked():
    print('Button clicked')

button = ttk.Button(root, text='Click Me', command=button_clicked)

button.pack()
root.mainloop()
```

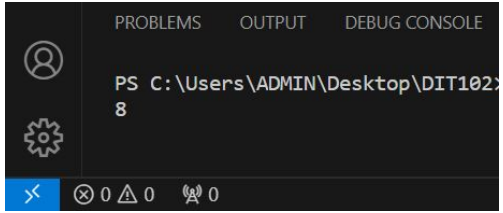
Define and Assign must have the same function name.

This program shows how to get values from a text box and calculate the value 2 to the power of n.

```
import tkinter as tk
from tkinter import ttk
root = tk.Tk()
root.geometry('300x200')
root.title('Button Widget')
tk.Label(root, text='Entry N').pack()
textbox = ttk.Entry(root,)
textbox.pack()

def button_clicked():
    print(  pow(2,  int(textbox.get()) ) )

button = ttk.Button(root, text='2 Power N', command=button_clicked)

button.pack()
root.mainloop()
```




$pow(2,n) = 2^n$



# Exercise

Write a program to calculate BMI from height, weight using Entries and Button. Make sure to show the result of BMI on the window.

# Tkinter button command arguments

If you want to pass the arguments to a callback function, you can use a lambda expression.

```
import tkinter as tk
from tkinter import ttk
root = tk.Tk()

def callback(param):
    print(param)

button = ttk.Button(root,text='Click Me',command=lambda:callback(args))

button.pack()
root.mainloop()
```

param = args

# Layout Management

## DIT102 Programming II

Asst. Prof. Dr. Parkpoom Chaisiriprasert

# Layout Management

Tkinter uses the geometry manager to organize widgets on a window. Tkinter supports three geometry managers:

- The pack geometry manager organizes widgets in blocks before placing them on the container widget, which can be the main window or a frame.
- The grid geometry manager uses the concepts of rows and columns to arrange the widgets.
- The placer geometry management gives you fine control over the positioning of widgets by allowing to Specify coordinates (x, y) and use relative positioning based on anchor points.

# Pack geometry

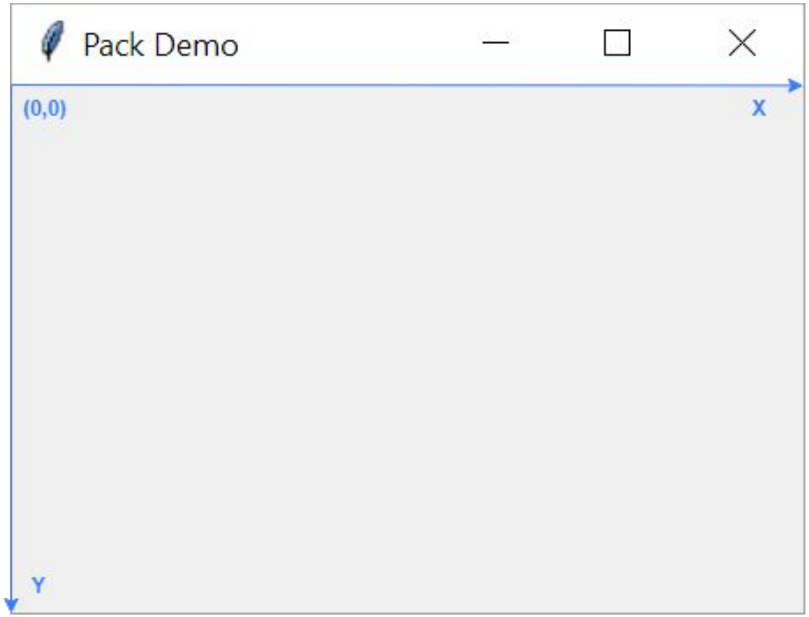
The pack geometry manager organizes widgets in blocks before placing them on the container widget, which can be the main window or a frame.

The pack geometry manager has many options that arrange the widgets:

- Side
- Expand
- Fill
- ipadx, ipady
- padx, pady
- Anchor

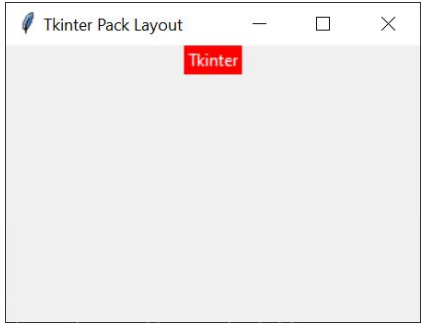
# Layout Management

To understand the x and y coordinates of the window: The top left corner of the window is the origin with the coordinate (0,0). The x-coordinate increments from left to right and the y-coordinate increments from top to bottom.



# The pack geometry

```
import tkinter as tk
root = tk.Tk()
root.title('Tkinter Pack Layout')
root.geometry('300x200')
label = tk.Label(master=root, text='Tkinter',bg='red',fg='white')
label.pack()
root.mainloop()
```




By default, the pack geometry manager places widgets in one direction vertically from top to bottom.

# Side Parameter

Side parameter determines the direction of the widgets in the pack layout.

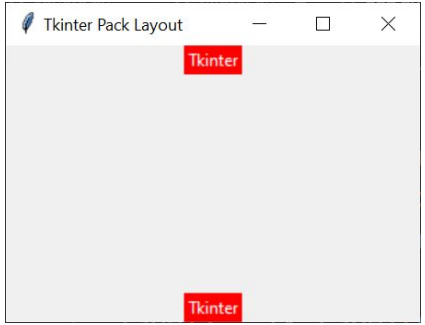
```
label.pack(side=tk.BOTTOM)
```

The side parameter has four options:

tk.TOP : arrange widgets from top to bottom vertically.
tk.BOTTOM : arrange widgets from bottom to top vertically.
tk.LEFT : arrange widgets from left to right horizontally.
tk.RIGHT : arrange widgets from right to left horizontally.

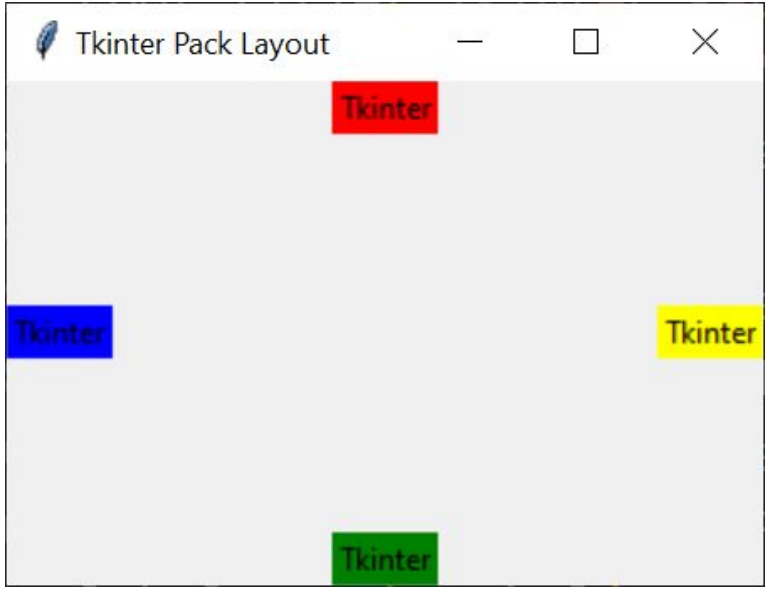
# Side Parameter

```
import tkinter as tk
root = tk.Tk()
root.title('Tkinter Pack Layout')
root.geometry('300x200')
label1 = tk.Label(master=root, text='Tkinter',bg='red',fg='white')
label1.pack()
label2 = tk.Label(master=root, text='Tkinter',bg='red',fg='white')
label2.pack(side=tk.BOTTOM)
root.mainloop()
```



# Exercise

Write a program to create a screen as shown in the example screenshot that follows.



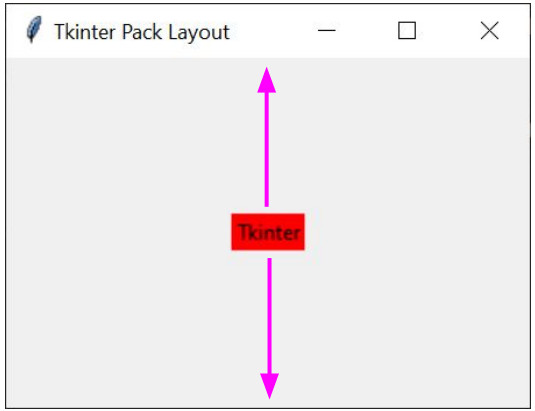
# Expand Parameter

The expand determines whether the widget should expand to occupy any extra spaces allocated to the container.

```
label.pack(expand=True)
```

The expand parameter of the Label to true, it occupies the entirely vertical from top to bottom.

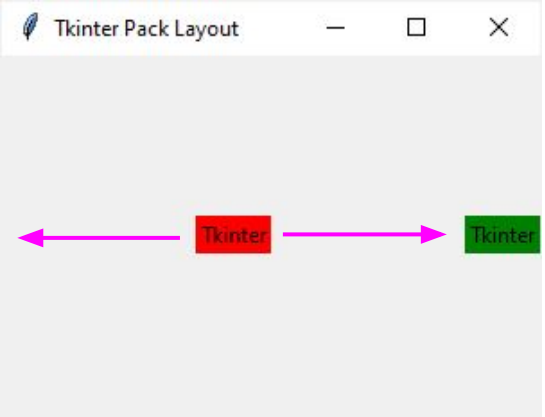
```
import tkinter as tk
root = tk.Tk()
root.title('Tkinter Pack Layout')
root.geometry('300x200')
label = tk.Label(master=root, text='Tkinter',bg='red')
label.pack(expand=True)
root.mainloop()
```



# Expand Parameter

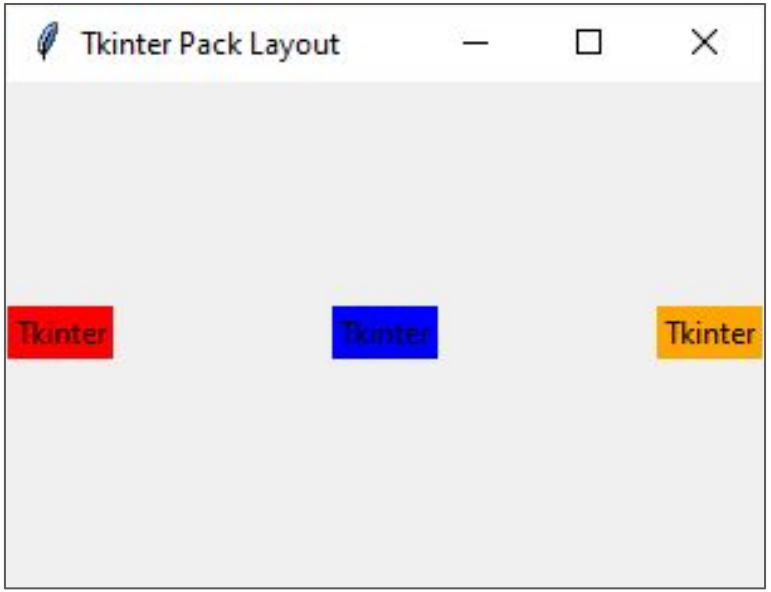
The expand parameter of the first Label to true, it occupies the entirely horizontal space and pushes the other Label widgets to the right:

```
import tkinter as tk
root = tk.Tk()
root.title('Tkinter Pack Layout')
root.geometry('300x200')
label = tk.Label(master=root, text='Tkinter',bg='red')
label.pack(side=tk.LEFT,expand=True)
label = tk.Label(master=root, text='Tkinter',bg='green')
label.pack(side=tk.LEFT)
root.mainloop()
```



# Exercise

Write a program to create a screen as shown in the example screenshot that follows.



# Fill Parameter

The fill determines if a widget will occupy the available space. By default, the fill is 'none'. The fill parameter has four options:

```
label.pack(fill=tk.X)
```

tk.NONE : The widget will not expand to fill any extra space. It only takes up space that fits the content.

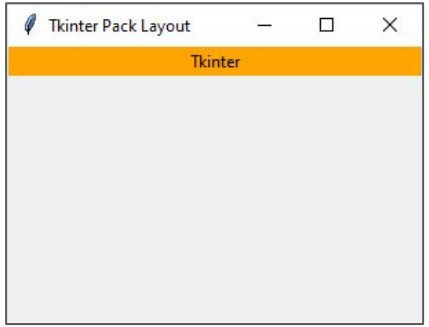
tk.X : The widget will expand horizontally to fill any extra space along the x-axis.

tk.Y : The widget will expand vertically to fill any extra space along the y-axis.

tk.BOTH : The widget will expand both horizontally and vertically to fill any extra space in both directions.

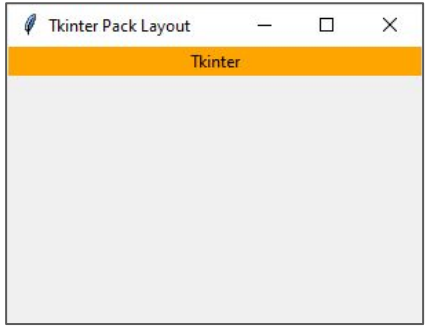
# Fill Parameter

```
import tkinter as tk
root = tk.Tk()
root.title('Tkinter Pack Layout')
root.geometry('300x200')
label = tk.Label(master=root, text='Tkinter',bg='red',fg='white')
label.pack(fill=tk.X)
root.mainloop()
```



# Fill Parameter

```
import tkinter as tk
root = tk.Tk()
root.title('Tkinter Pack Layout')
root.geometry('300x200')
label = tk.Label(master=root, text='Tkinter',bg='red',fg='white')
label.pack(fill=tk.X)
root.mainloop()
```



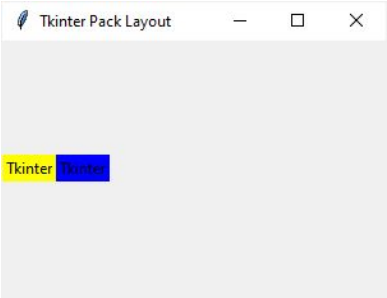
# Exercise

Write a program to create a screen as shown in the example screenshot that follows.

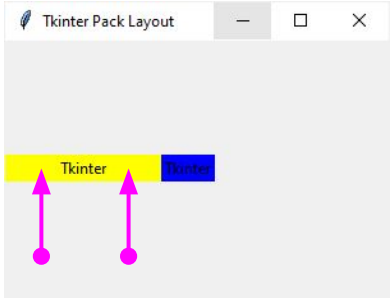
# Internal paddings: ipadx & ipady

The ipadx and ipady parameters create internal paddings for widgets:

- ipadx creates padding left and right, or padding along the x-axis.
- ipady creates padding top and bottom, or padding along the y-axis.



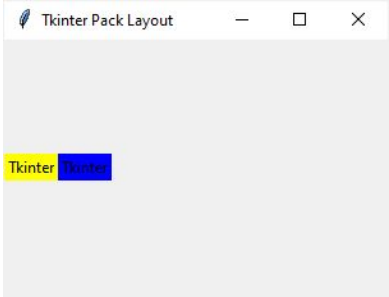

Without Internal padding



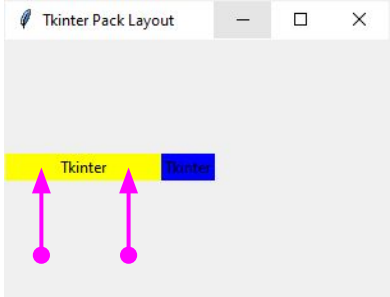

With Internal padding

# Internal paddings: ipadx & ipady

```
import tkinter as tk
root = tk.Tk()
root.title('Tkinter Pack Layout')
root.geometry('300x200')
label = tk.Label(master=root, text='Tkinter',bg='yellow')
label.pack(side=tk.LEFT, ipadx=40)
label = tk.Label(master=root, text='Tkinter',bg='blue')
label.pack(side=tk.LEFT)
root.mainloop()
```




Without Internal padding




With Internal padding

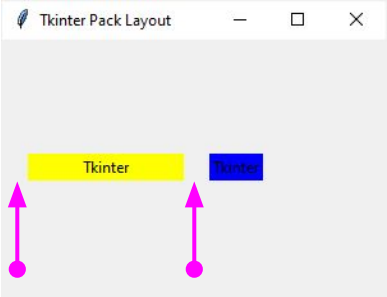
# Exercise

Write a program to create a screen as shown in the example screenshot that follows.

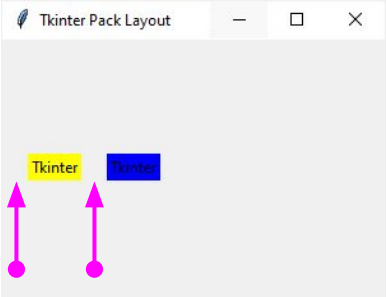
# External paddings: padx and pady

The padx and pady parameters create external paddings for widgets:

- padx – represents the horizontal padding that adds space to the left and right of the widget.
- pady – represents the vertical padding that adds space above or below the widget.



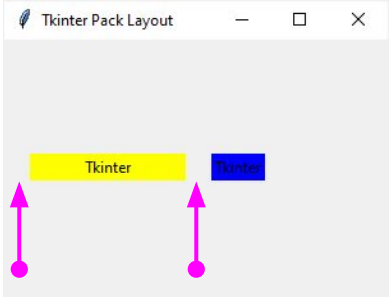

With External and Internal padding



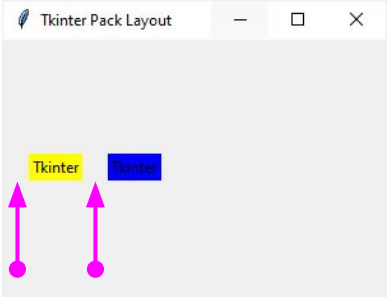

Without Internal padding

# External paddings: padx & pady

```
import tkinter as tk
root = tk.Tk()
root.title('Tkinter Pack Layout')
root.geometry('300x200')
label = tk.Label(master=root, text='Tkinter',bg='yellow')
label.pack(side=tk.LEFT, ipadx=40, padx=20)
label = tk.Label(master=root, text='Tkinter',bg='blue')
label.pack(side=tk.LEFT)
root.mainloop()
```




With External and Internal padding




Without Internal padding

# Exercise

Write a program to create a screen as shown in the example screenshot that follows.

# Anchor Parameter

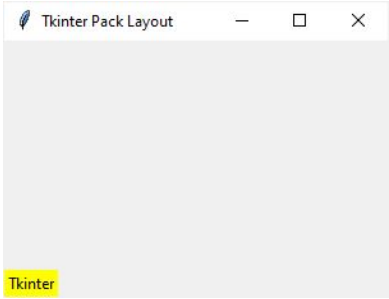
The anchor parameter allows you to anchor the widget to the edge of the allocated space. It accepts one of the following values:

```
label.pack(anchor=tk.W)
```

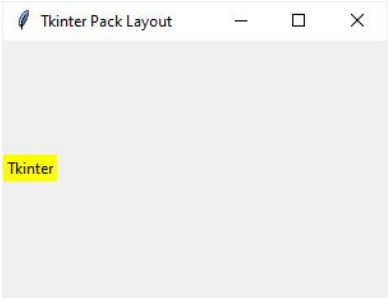
tk.N : North or Top Center
tk.E : East or Right Center
tk.S : South or Bottom Center
tk.W : West or Left Center

# Anchor Parameter

```
import tkinter as tk
root = tk.Tk()
root.title('Tkinter Pack Layout')
root.geometry('300x200')
label = tk.Label(master=root, text='Tkinter',bg='yellow')
label.pack(side=tk.LEFT, anchor=tk.S)
root.mainloop()
```



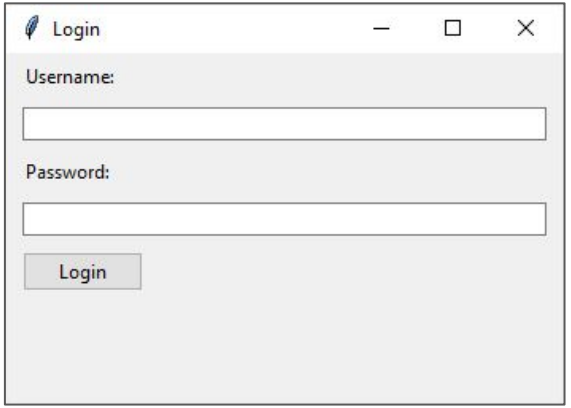

With side and anchor




Without anchor

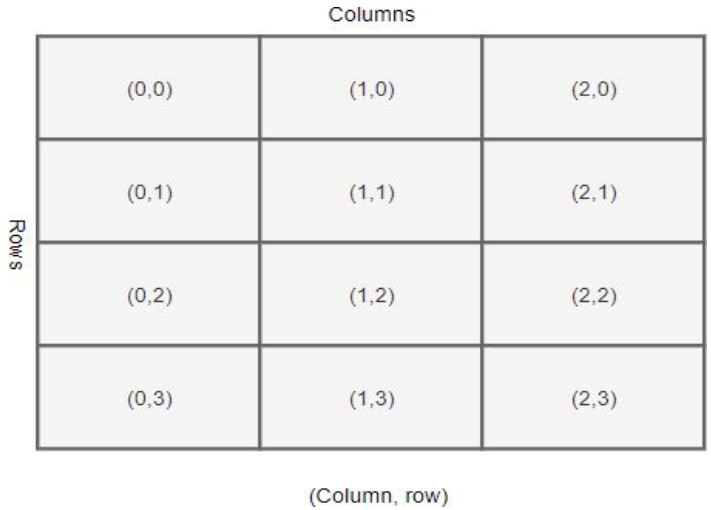
# Exercise

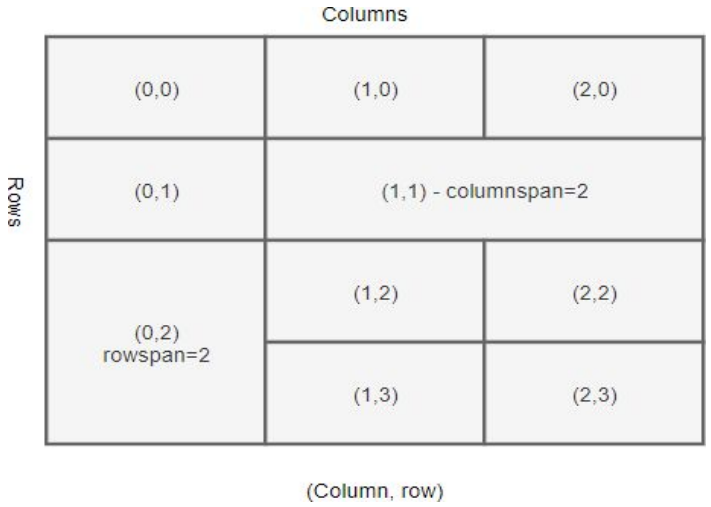
Write a program to create a screen as shown in the example screenshot that follows.



# Tkinter Grid

The grid geometry manager uses the concepts of rows and columns to arrange the widgets.




Rows and columns can span. The following illustrates a grid that has the cell (1,1) that spans two columns and the cell (0,2) that spans two rows:

# Setting up the grid

To configure the rows and columns of the grid

The columnconfigure() method configures the column index of a grid.

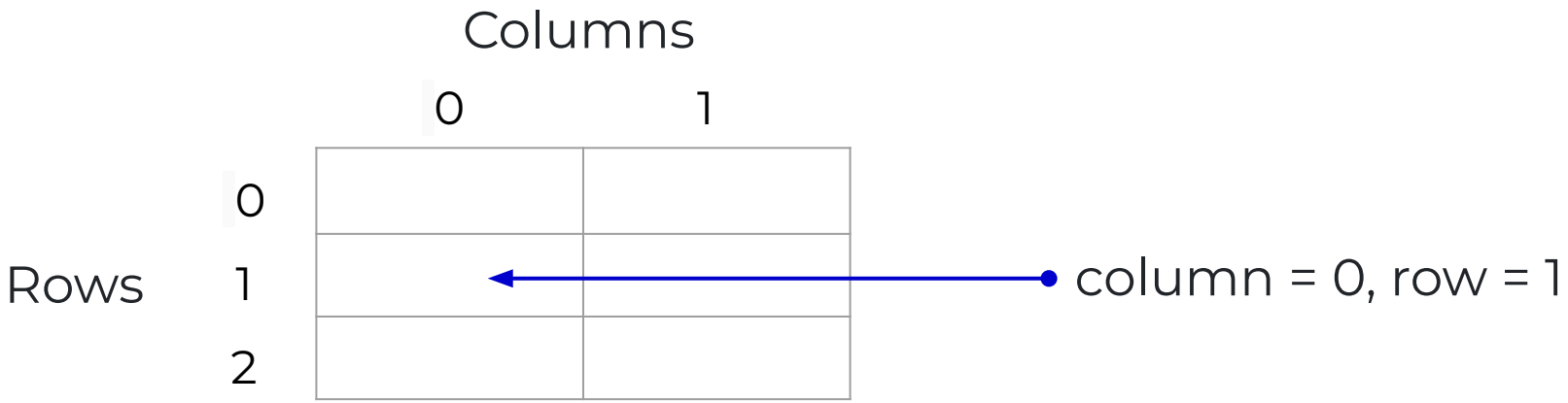
```
container.columnconfigure(index, weight)
```

The weight determines how wide the column will occupy, which is relative to other columns.

# Setting up the grid

A grid that has two columns and three rows.

```
container.columnconfigure(0, 1)
container.columnconfigure(1, 3)
```



# Positioning a widget on the grid

To place a widget on the grid, you use the widget's grid() method:

```
widget.grid(**options)
```

**The grid() method has the following parameters:**

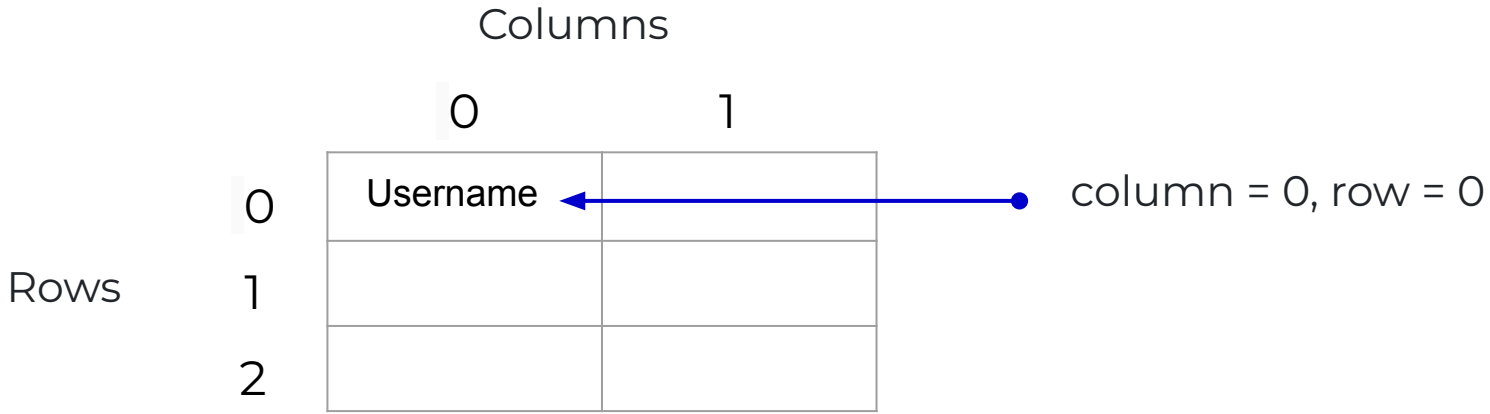
column : The column index where you want to place the widget.
row : The row index where you want to place the widget.
rowspan : Set the number of adjacent rows that the widget can span.
columnspan : Set the number of adjacent columns that the widget can span.
sticky : specifies which side the widget should stick to and how to distribute any extra space within the cell

Also used : padx, pady, ipadx and ipady

# Positioning a widget on the grid

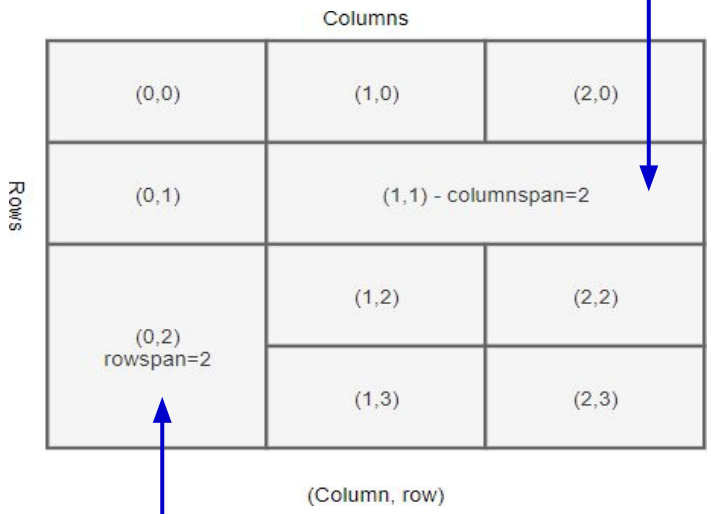
A grid that has two columns and three rows.

```
username_label = ttk.Label(root, text="Username:")
username_label.grid(column=0, row=0)
```



# Columnspan

```
label.grid(column=1, row=1, columnspan=2)
```

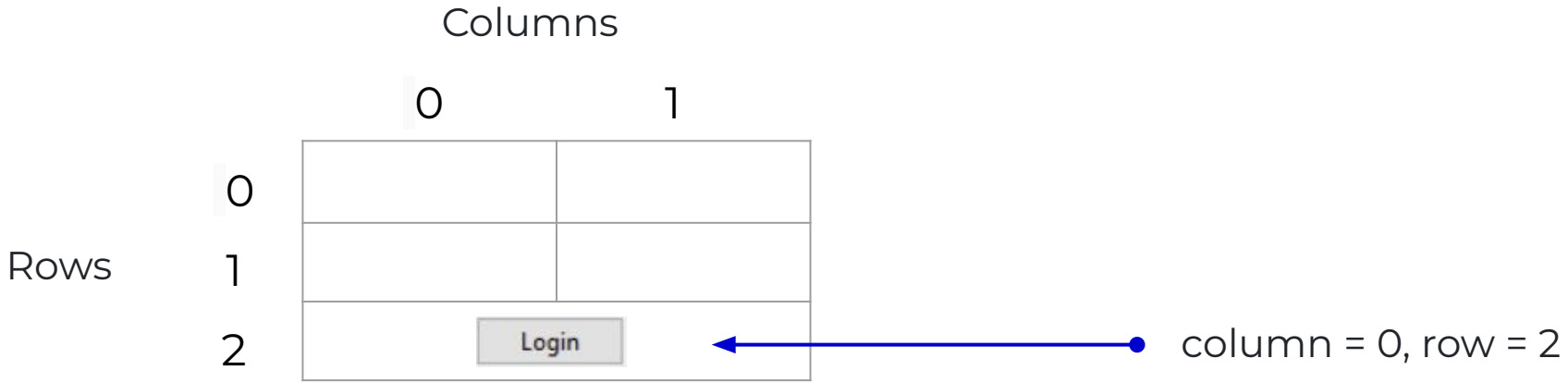



```
label.grid(column=0, row=2, rowspan=2)
```

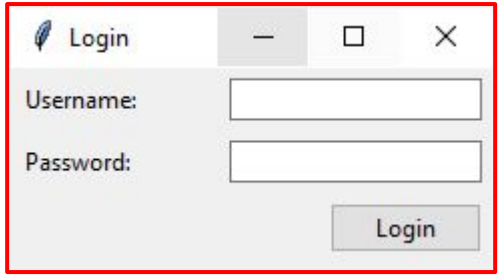
# Columnspan

Set the number of adjacent columns that the widget can span.

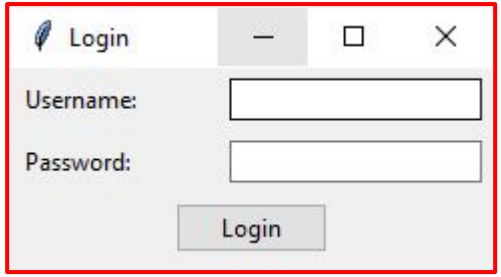
```
login_button = ttk.Button(root, text="Login")
login_button.grid(column=0, row=2, columnspan=2)
```



```
import tkinter as tk
from tkinter import ttk
root = tk.Tk()
root.geometry("240x100")
root.title('Login')
root.columnconfigure(0, weight=1)
root.columnconfigure(1, weight=3)
username_label = ttk.Label(root, text="Username:")
username_label.grid(column=0, row=0, sticky=tk.W, padx=5, pady=5)
username_entry = ttk.Entry(root)
username_entry.grid(column=1, row=0, sticky=tk.E, padx=5, pady=5)
password_label = ttk.Label(root, text="Password:")
password_label.grid(column=0, row=1, sticky=tk.W, padx=5, pady=5)
password_entry = ttk.Entry(root,  show="*")
password_entry.grid(column=1, row=1, sticky=tk.E, padx=5, pady=5)
login_button = ttk.Button(root, text="Login")
login_button.grid(column=1, row=2, sticky=tk.E, padx=5, pady=5)
root.mainloop()
```



```
import tkinter as tk
from tkinter import ttk
root = tk.Tk()
root.geometry("240x100")
root.title('Login')
root.columnconfigure(0, weight=1)
root.columnconfigure(1, weight=3)
username_label = ttk.Label(root, text="Username:")
username_label.grid(column=0, row=0, sticky=tk.W, padx=5, pady=5)
username_entry = ttk.Entry(root)
username_entry.grid(column=1, row=0, sticky=tk.E, padx=5, pady=5)
password_label = ttk.Label(root, text="Password:")
password_label.grid(column=0, row=1, sticky=tk.W, padx=5, pady=5)
password_entry = ttk.Entry(root,  show="*")
password_entry.grid(column=1, row=1, sticky=tk.E, padx=5, pady=5)
login_button = ttk.Button(root, text="Login")
login_button.grid(column=0, row=2, columnspan=2, padx=5, pady=5)
root.mainloop()
```



# End.

DIT102