GUI Using Python

Features to explore:

- ✓Graphical User Interfaces
- ✓Using the `tkinter` Module
- ✓Using Text with `Label` Widgets
- ✓Organizing Widgets with Frames
- ✓`Button` Widgets and Info Dialog Boxes
- ✓Getting Input with the `Entry` Widget
- ✓Using Labels as Output Fields
- ✓Radio Buttons and Check Buttons
- ✓Drawing Shapes with the `Canvas` Widget

Graphical User Interfaces

- **User Interface**: the part of the computer with which the user interacts
- Command line interface: displays a prompt and the user types a command which is then executed
- **Graphical User Interface (GUI):** allows users to interact with a program through graphical elements on the screen

GUI Programs Are Event-Driven

- **In text-based environments:**
- programs determine the order in which things happen
  - The user can only enter data in the order as per the program
- **GUI environment is event-driven:**
  - The user determines the order in which things happen
    - User causes events to take place and the program responds to the events

- No GUI programming features built into Python
- `tkinter` **module:** allows to create simple GUI programs
  - Comes with Python
- **Widget:** graphical element that the user can interact with or view
  - Presented by a GUI program

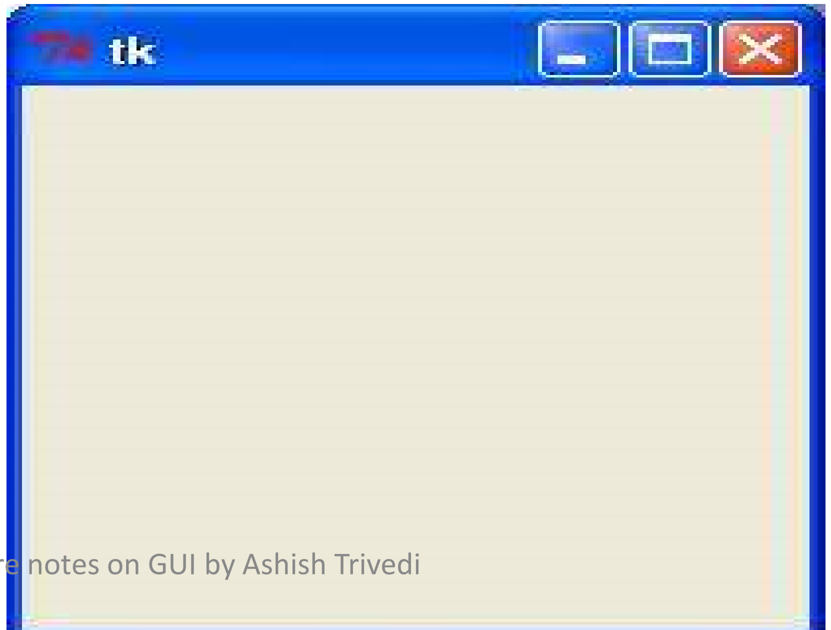
- We take an object-oriented approach when writing GUI programs
  - `__init__` method builds the GUI
  - When an instance is created the GUI appears on the screen

Applications where the GUI is utilized are:

- Creating a Calculator which would have a user-interface and functionalities which are there in a calculator.
- Text-Editors, IDE's .
- Sudoku, Chess, Solitaire, etc.., are games that you can play using GUI apps.
- Chrome, Firefox, Microsoft Edge, etc. used to surf the internet is a GUI app.
- A GUI for controlling a Drone from your laptop, and through the buttons we can control the Drone through screen which would show the camera feed captured by the Drone in a real-time.

- Tkinter is the standard GUI library for Python.
- Python when combined with Tkinter provides a fast and easy way to create GUI applications.
- Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.
- Creating a GUI application using Tkinter requires to perform the following steps –
    - Import the *Tkinter* module.
    - Create the GUI application main window.
    - Adding widgets to the GUI application.
    - Enter the main event loop to take action against each event triggered by the user.

```
from Tkinter import *
window = Tk()
# Code to add widgets will go here...
window.mainloop()
```



Tkinter Widgets

- Tkinter provides various controls, such as buttons, labels and text boxes used in a GUI application. These controls are called widgets.
- tkinter offers a method :Tk(screenName=None, baseName=None, className='Tk', useTk=1): To create a main window.
- window=tkinter.Tk() where window is the name of the main window object
- mainloop(): is used when an application is ready to run.
- mainloop() is an infinite loop used to run the application, wait for an event to occur and process the event as long as the window is not closed.
- Executed using:window.mainloop()

First, import the Tkinter module.

- Next initialize the window manager with the tkinter.Tk() method and assign it to a variable. This method creates a blank window with close, maximize, and minimize buttons on the top.

- Then if required can Rename the title of the window using window.title(title_of_the_window).
- Next, can use a widget called Label, which is used to insert some text into the window.

- Then, can make use of Tkinter's geometry management attribute called pack() to display the widget in size it requires.

- Finally, use the mainloop() method to display the window until you manually close it. It runs an infinite loop in the backend.

- Widgets are like elements in HTML.
- There are different types of widgets for different types of elements in the Tkinter.
- These are standard GUI elements and provide the user with controls like buttons, text, menus, and text boxes.
- *These sub components or controls of Graphical User Interface (GUI) are known as **widgets** in Tkinter.*

- In **Tkinter** , **Widgets** are objects which are instances of classes representing buttons, frames and others.
- Each widget is a Python object.
- When creating a widget, we need to pass its parent as a parameter to the widget creation function.
- The only exception is the “root” window, which is the top-level window that will contain everything else and it does not have a parent.

For Eg.

```
top=Tk()
Widget1=Widget(Root)
```

- `Label` **widget**: displays a single line of text in a window
  - Made by creating an instance of `tkinter` module's `Label` class
  - Format:
    ```
    tkinter.Label(self.main_window,
                  text = 'my text')
    ```
    - First argument references the root widget,
    - second argument shows **text that should appear in label**

Geometry Management

tkinter provides access to the geometric configuration of the widgets which can organize the widgets in the parent windows.

There are mainly three geometry manager classes .

1.**pack() method:**It organizes the widgets in blocks before placing in the parent widget.

2.**grid() method:**It organizes the widgets in grid (table-like structure) before placing in the parent widget.

3.**place() method:**It organizes the widgets by placing them on specific positions specified by the programmer.

- For organizing all the widgets in the parent window, Tkinter provides the geometric configuration of the widgets.
- The GUI Application Layout is controlled by Geometric Managers of Tkinter.
- **Each window and Frame in application can use only one geometry manager.**
- But, different frames can use different geometry managers, even if they're already assigned to a frame or window using another geometry manager.

- `pack` method: determines where a widget should be positioned and makes it visible when the main window is displayed
  - Called for each widget in a window
  - Receives an argument to specify positioning
    - Positioning depends on the order in which widgets were added to the main window
    - arguments: `side='TOP', side='LEFT', side='RIGHT'`

Packing Algorithm:

It compute a rectangular area called Parcel which is tall (or wide) enough to hold the widget and then it will fill the remaining width (or height) in the window with blank space.

Next it will center the widget until any different location is specified.

syntax for using pack() function:
widget.pack(options)

- fill

The default value of this option is set to NONE. Also, we can set it to X or Y in order to determine whether the widget contains any extra space. [fill=BOTH]

- side

This option specifies which side to pack the widget against.
TOP : If we want to pack widgets vertically [Default]
LEFT: To pack horizontally

- expand

This option is used to specify whether the widgets should be expanded to fill any extra space in the geometry master or not. Its default value is false. If it is false then the widget is not expanded otherwise widget expands to fill extra space.[expand=True]

- The padx and pady attributes add extra horizontal and vertical space to the widgets.

- `Button` widget: widget that the user can click to cause an action to take place
  - When creating a button can specify:
    - Text to appear on the button
    - A callback function
- Callback function: function or method that executes when the user clicks the button
  - Also known as an event handler

- Info dialog box: a dialog box that shows information to the user
  - Format for creating an info dialog box:
    - Import `tkinter.messagebox` module
    - `tkinter.messagebox.showinfo(`*`title`*`,`
      `message``)`
      - *`title`* is displayed in dialog box's title bar
      - *`message`* is an informational string displayed in the main part of the dialog box

- Quit button: closes the program when the user clicks it
- To create a quit button in Python:
  - Create a `Button` widget
  - Set the root widget's `destroy` method as the callback function
    - When the user clicks the button the `destroy` method is called and the program ends

Tkinter reliefs

A relief is a border decoration. The values are:

SUNKEN, RAISED, GROOVE, RIDGE, and FLAT.

Tkinter cursors

The cursor is a small icon that shows where the mouse pointer is

located. The cursor in Tkinter is set with the cursor attribute.

tcross ,heart, pencil,hand2

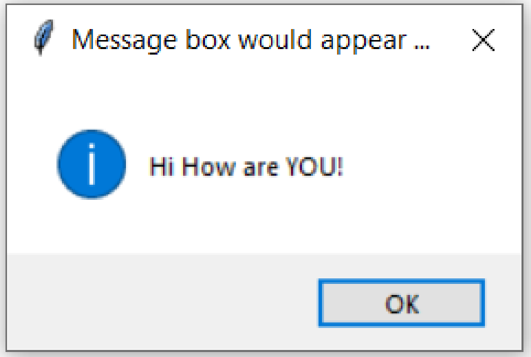
Here is list of possible constants which can be used for relief attribute.

- FLAT
- RAISED
- SUNKEN
- GROOVE
- RIDGE

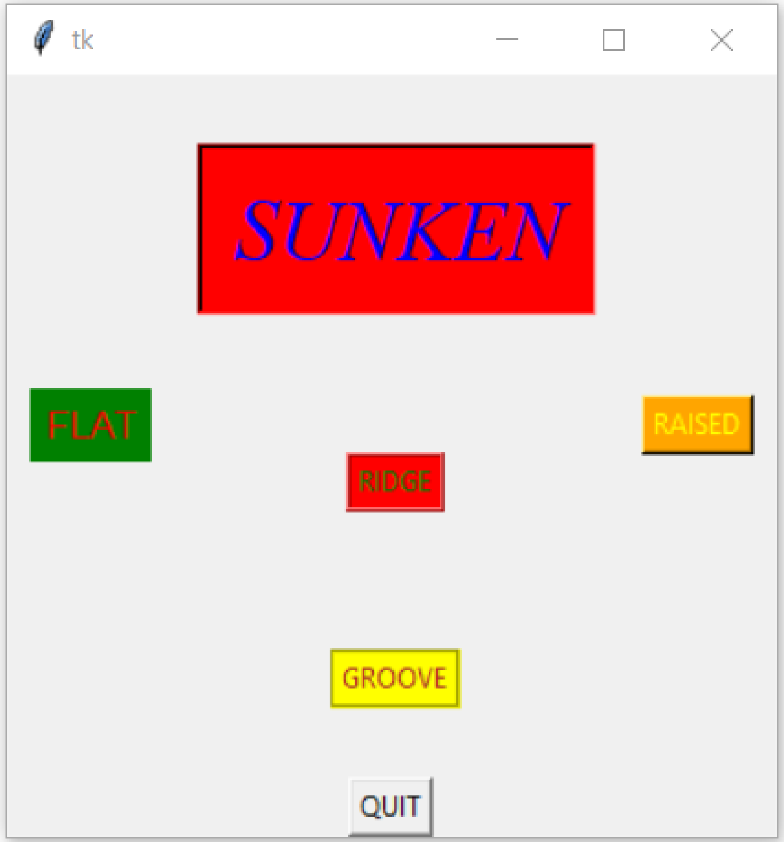
Example

```
from Tkinter import *
import Tkinter

top = Tkinter.Tk()

B1 = Tkinter.Button(top, text ="FLAT", relief=FLAT )
B2 = Tkinter.Button(top, text ="RAISED", relief=RAISED )
B3 = Tkinter.Button(top, text ="SUNKEN", relief=SUNKEN )
B4 = Tkinter.Button(top, text ="GROOVE", relief=GROOVE )
B5 = Tkinter.Button(top, text ="RIDGE", relief=RIDGE )

B1.pack()
B2.pack()
B3.pack()
B4.pack()
B5.pack()
```

Hello ! Enjoying learing GUI features


Message box would appear ...

Hi How are YOU!



Button widget State:

I. NORMAL

II. ACTIVE

III. DISABLED

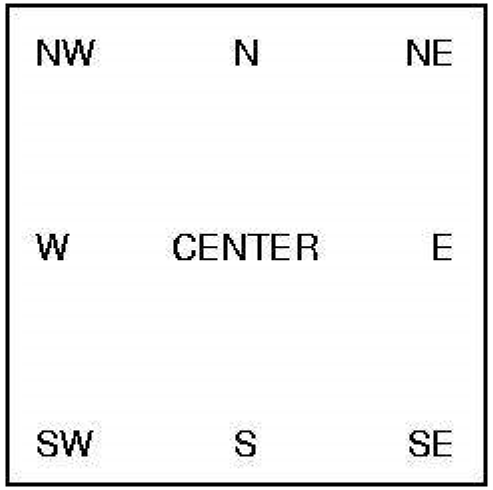
Syntax:

```
Button_obj = Button(root, options)
```

```
from tkinter import *

root_window = Tk()

root_window.geometry("400x400")

button_obj = Button(root_window,text = "Computer Science")

button_obj.pack()

root_window.mainloop()
```

Anchors represents text positioning relative to a reference point.

if we choose CENTER as a text anchor, the text will be positioned horizontally and vertically w.r.t. the reference point.



Radiobuttons represent either text or images, and is associated with a function or method with each button. When button object is selected, Tkinter fetches corresponding function or method.

Syntax:

Button_obj = Radiobutton(master, text=” Button title”, variable = “shared variable”, value = “values of each button”, options = values, …)

- shared variable = variable is shared among all Radio buttons
- value = each radiobutton have different value otherwise more than 1 radiobutton will get selected.

Checkbutton

- ✓ Used to implement the Boolean choice :on/off selections.
- ✓ Provides multiple choices to the user from which, the user needs to select one.
- ✓ Exhibits many of many selections.
- ✓ Uses the <input> tag in HTML, to create checkbox

syntax :

```
Check_button_object = checkbutton(parent_window,
                                          option=value)
```

offvalue	The **associated control variable** of checkbutton is set to 0 by default if the button is (off).
onvalue	The **associated control variable of checkbutton** will be set to 1 when it is set (on).
variable	used to represents the associated variable that **is used to track the state of the checkbutton**
state	**used to represent the state of the checkbutton**. **Its default value= normal**. It can be changed to DISABLED to make the checkbutton unresponsive. The value of this button is ACTIVE when checkbutton is under focus

Python - Tkinter Frame

- The Frame widget is used for the process of grouping and organizing other widgets.
- It acts like a container, which is responsible for arranging the position of other widgets.
- It uses rectangular areas in the screen to organize the layout and to provide padding of these widgets.
- A frame can also be used as a foundation class to implement complex widgets.

Syntax:

```
frame_obj = Frame ( root, option, ... )
```

Parameters:

root: This represents the parent window.

options: These can be used as key-value pairs separated by commas.

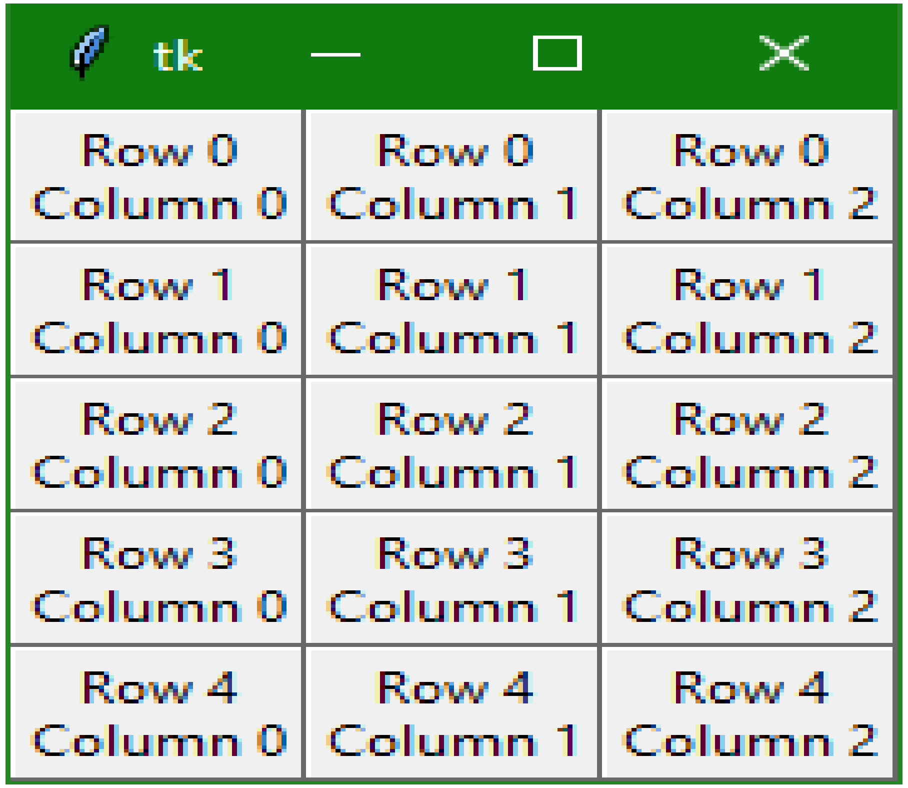
Grid Manager

- The Grid geometry manager places the widgets in a 2-dimensional table, which consists of a number of rows and columns.
- The position of a widget is defined by a row and a column number.
- Widgets with the same column number and different row numbers will be above or below each other.
- Correspondingly, widgets with the same row number but different column numbers will be on the same "line" and will be beside of each other, i.e. to the left or the right.

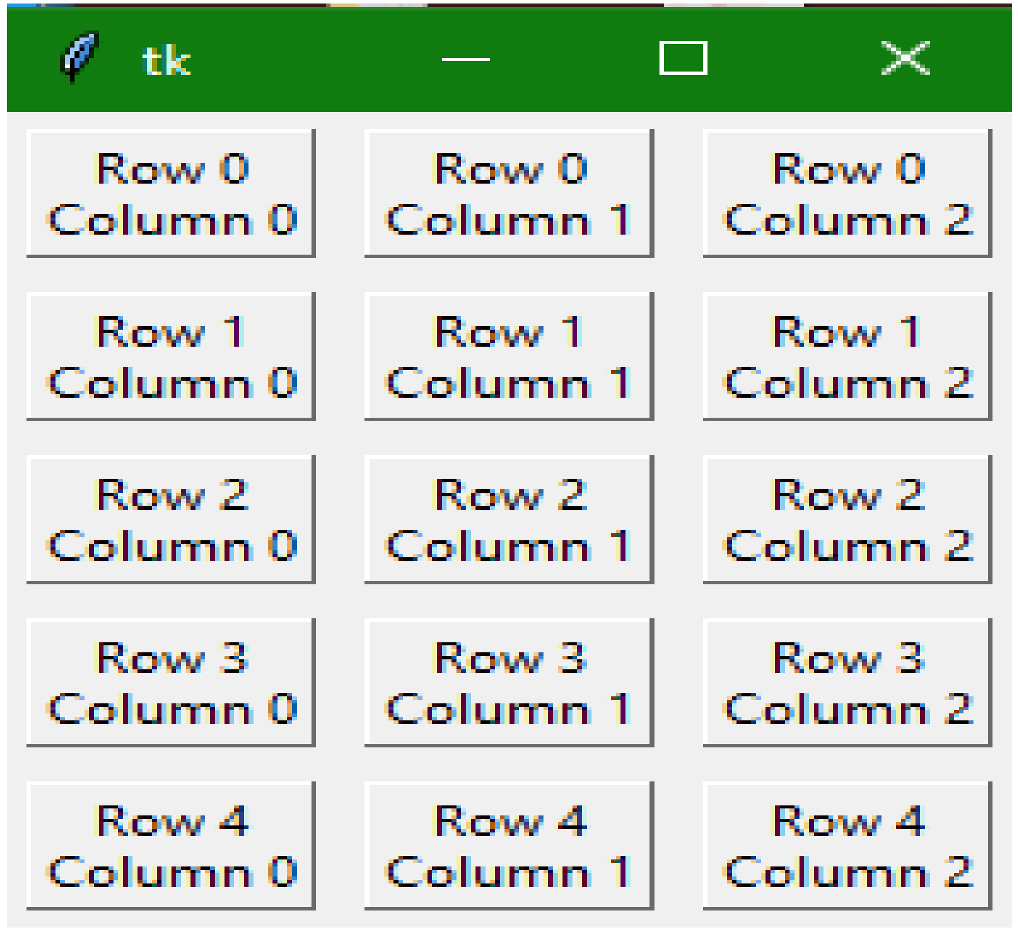
- The size of the grid doesn't have to be defined, because the manager automatically determines the best dimensions for the widgets used.

- **column:** This option is used to put the widget in a column that is leftmost column. The default column is 0.
- **columnspan:** This option keeps the track of how many columns it will take to occupy widgets and by default, this is 1.
- **ipadx and ipady:** These two options are used for how many pixels on the interface to pad the widgets in horizontal and vertical respectively but it will be used in padding inside widgets border.
- **padx and pady:** These two options are similar to the above options but these are used to pad outside widget borders in horizontal and vertical padding the pixels.

- **row:** This option is when the widget is to put in a row, by default the first row is empty.
- **rowspan:** This option will be used to tell how many row widgets are occupied and the default value is 1.
- **sticky:** This option is used when a cell cannot fit in the widget which means when the cell is larger than widget then this option is used to know which sides and corners of the widgets the cell can stick to. By default, widgets are always centered in the cell. Sticky uses similar to compass directions to stick the cells to the widgets like North, south, east, west and all the combination of these four.



Without Padding



With Padding

Entry widget:

- The Entry widget is used to accept single-line text strings from user.
- For displaying multiple lines of text which can be edited, *Text* widget should be used.
- For display one or more lines of text that cannot be modified by the user, *Label* widget should be used.

Syntax:

```
            entry_object = Entry( rootwindow,
option, ... )
```

Parameters:

rootwindow: This represents the parent window.

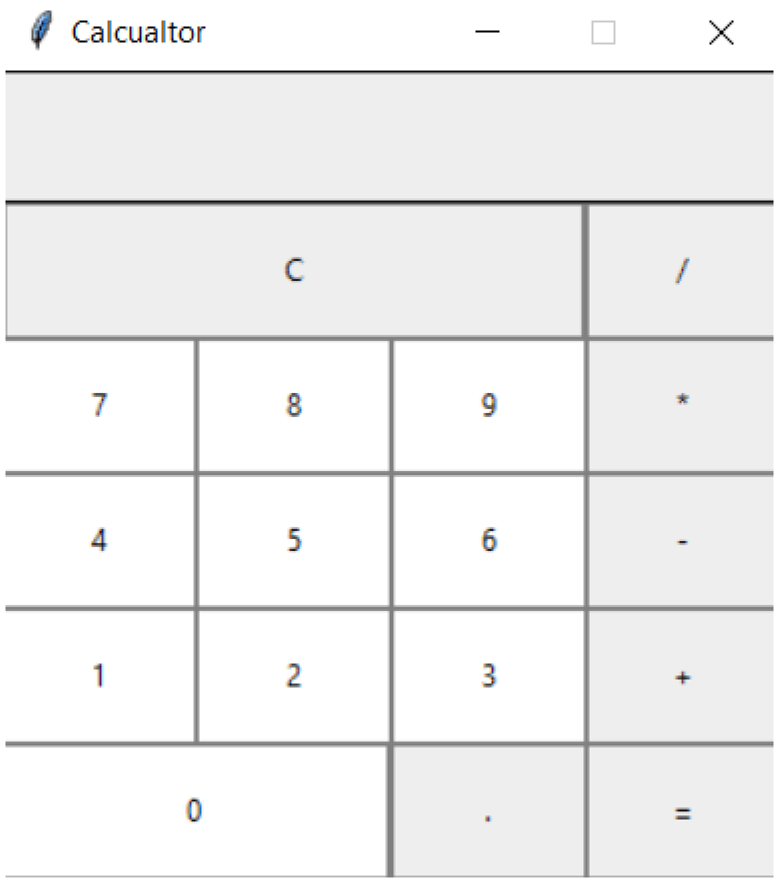
options: used as key-value pairs separated by commas. B,

Python Packages for Data Science/Big Data/Machine Learning:

- **pandas** (data analysis)
- **NumPy** (multi-dimensional arrays)
- **SciPy** (algorithms to use with numpy)
- **HDF5** (store & manipulate data)
- **Matplotlib** (data visualization)
- **Jupyter** (research collaboration)
- **PyTables** (managing HDF5 datasets)
- **HDFS** (C/C++ wrapper for Hadoop)
- **pymongo** (MongoDB driver)

- **SQLAlchemy** (Python SQL Toolkit)

Assignment: Constructing calculator



Listbox :

used to display a list of entries for multiple selection

Syntax

listbox_obj= Listbox (master, option, ...)

Parameters:

master – This represents the parent window.

options – These options can be used as key-value pairs separated by commas.

selectmode

Helps in entry selection

- **BROWSE –** select one line out of a listbox. If click on an entry and then drag to a different line, the selection will follow the mouse. This is the default.
- **SINGLE –** select one line, and you can't drag the mouse. wherever you click button 1, that line is selected.
- **MULTIPLE** – select any number of lines at once. Clicking on any line toggles: whether or not it is selected.
- **EXTENDED** – select any adjacent group of lines at once by clicking on the first line and dragging to the last line

Text widget:

Used for multi-line text area.
Text widgets can also be used as simple text editors or as a web browsers.

The Text widget is used to provide the text editor to the user.

Syntax

```
textobject = Text(root, options)
```

insert(index, string)	It is used to insert the specified string at the given index.

wrap

This option controls the display of lines that are too wide. Set wrap=WORD and it will break the line after the last word that will fit. With the default behavior, wrap=CHAR, any line that gets too long will be broken at any character.

xscrollcommand

To make the text widget horizontally scrollable, set this option to the set() method of the horizontal scrollbar.

yscrollcommand

To make the text widget vertically scrollable, set this option to the set() method of the vertical scrollbar.

Drawing Shapes with the `Canvas` Widget

- The `Canvas` widget is a blank, rectangular area that allows to draw 2D shapes.

- `Canvas` widget's *screen coordinate system* is used to specify the location of graphics.

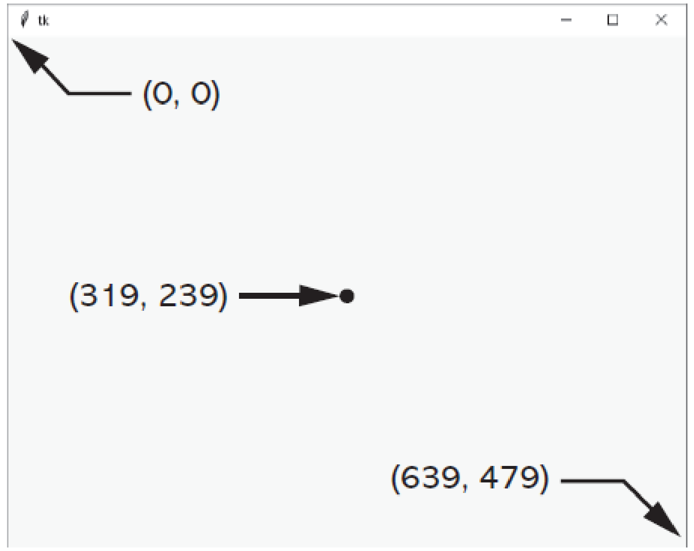
- The coordinates of the pixel in the upper-left corner of the screen are (0, 0).
  - The *X* coordinates increase from left to right
  - The *Y* coordinates increase from top to bottom.
- We can also display various custom widgets .

Syntax:

Canvas_object = Canvas(root, height, width, bd, bg, ..)

Drawing Shapes with the `Canvas` Widget

pixel locations in a 640 by 480 window



Drawing Shapes with the `Canvas` Widget

- The `Canvas` widget has different methods for drawing graphical shapes on the surface of the widget.
- The methods are:
  - `create_line`
  - `create_rectangle`
  - `create_oval`
  - `create_arc`
  - `create_polygon`
  - `create_text`

Drawing a Line

$$\texttt{\textit{canvas\_name}.create\_line(}\underbrace{\texttt{x1, y1,}}_{\text{Coordinates of the line's starting point}}\ \overbrace{\texttt{x2, y2,}}^{\text{Coordinates of the line's ending point}}\ \underset{\text{Optional arguments}}{\texttt{\textit{options}...}}\texttt{)}$$

Drawing a Rectangle

$$\textit{canvas\_name}\texttt{.create\_rectangle(}\underbrace{\texttt{x1, y1,}}_{\text{Coordinates of the upper-left corner}}\ \overbrace{\texttt{x2, y2,}}^{\text{Coordinates of the lower-right corner}}\ \underset{\text{Optional arguments}}{\textit{options…}}\texttt{)}$$

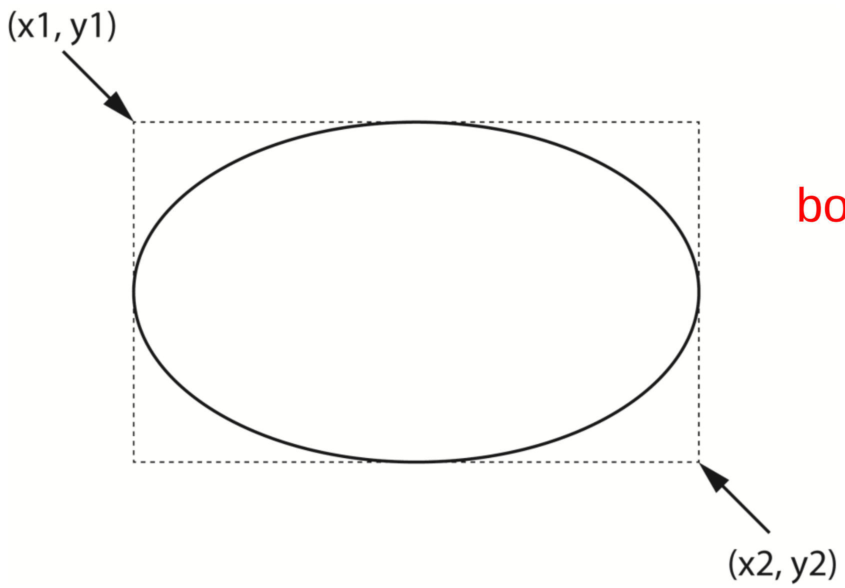
Drawing an Oval

Coordinates of
the lower-right
corner of
bounding rectangle

```
canvas_name.create_oval(x1, y1, x2, y2, options…)
```

Coordinates of
the upper-left
corner of
bounding rectangle

Optional arguments



Drawing an Arc

Coordinates of the upper-left corner of bounding rectangle

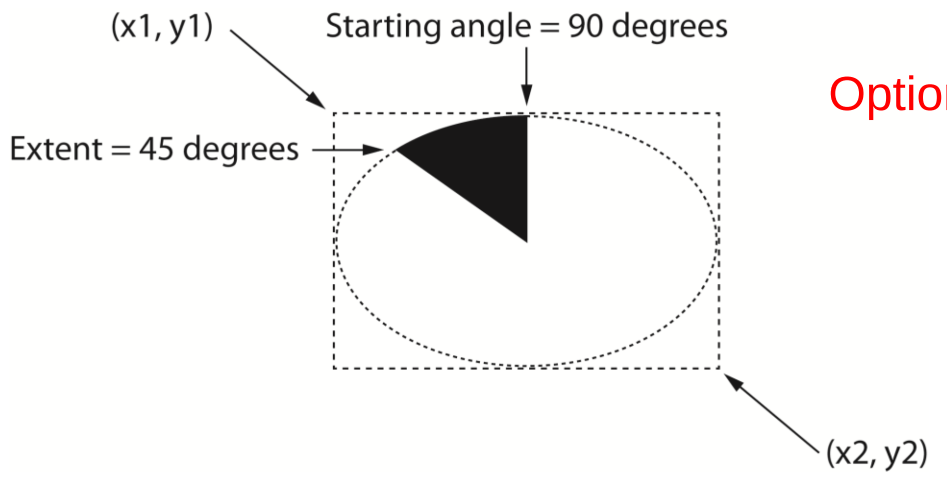
Coordinates of the lower-right corner of bounding rectangle

```
canvas_name.create_arc(x1, y1, x2, y2,
                       start=angle, extent=width,
                       options…)
```

Starting angle

Counter clockwise extent of the arc

Optional arguments

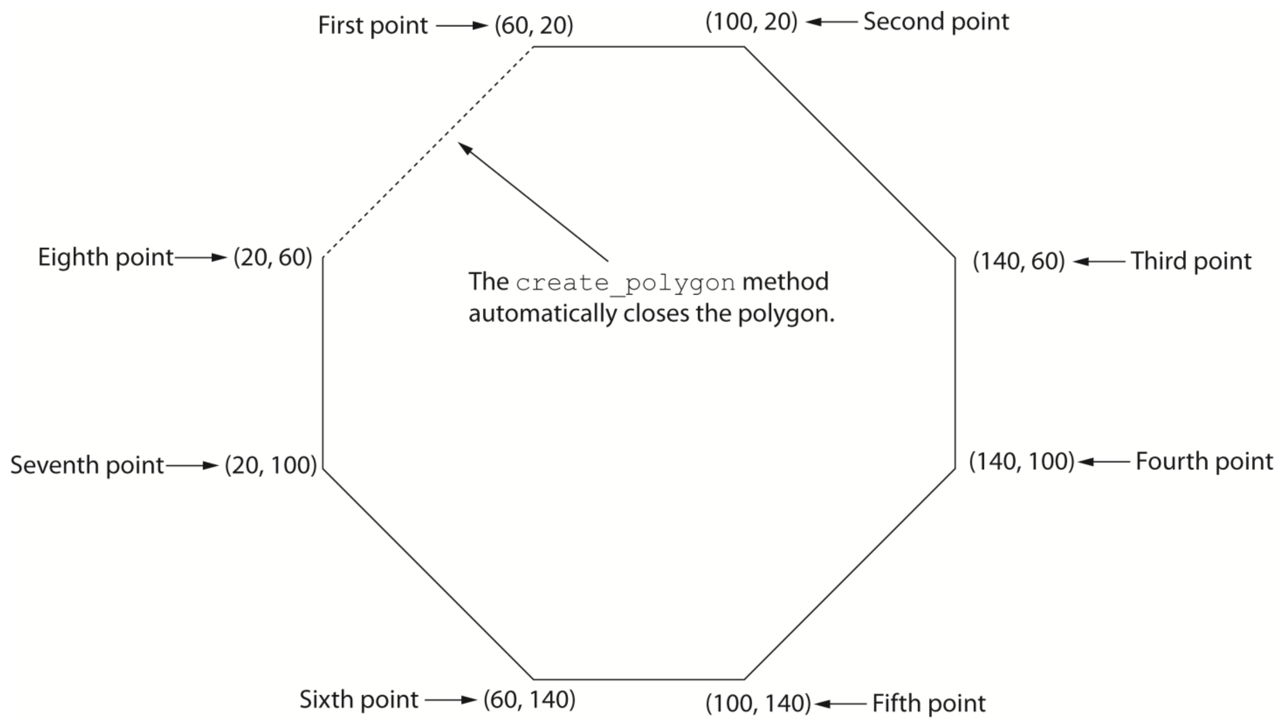


Drawing a Polygon

$$\texttt{\textit{canvas\_name}.create\_polygon(}\underbrace{\texttt{x1, y1,}}_{\text{Coordinates of the first vertex}}\ \overbrace{\texttt{x2, y2,}}^{\text{Coordinates of the second vertex}}\ \texttt{…,}\underbrace{\texttt{\textit{options}…}}_{\text{Optional arguments}}\texttt{)}$$

Drawing a Polygon

```
self.canvas.create_polygon(60, 20, 100, 20, 140, 60, 140, 100,
                           100, 140, 60, 140, 20, 100, 20, 60)
```



Displaying Text on the Canvas

Text to display

```
canvas_name.create_text(x, y, text=text, options…)
```

Coordinates of the text's insertion point

Optional arguments

- **root** = root window.
- **height** = height of the canvas widget.
- **width** = width of the canvas widget.
- **bg** = background colour for canvas.
- **bd** = border of the canvas window.
- **scrollregion** (w, n, e, s)tuple defined as a region for scrolling left, top, bottom and right
- **highlightcolor** colour shown in the focus highlight.
- **cursor** It can defind as a cursor for the canvas which can be a circle, an arrow etc.
- **confine** decides if canvas can be accessed outside the scroll region.
- **relief** type of the border which can be SUNKEN, RAISED, GROOVE and RIDGE.

The binding function is used to deal with the events.

We can bind **Python's Functions** and methods to an event as well as we

can bind these functions to any particular widget.

Syntax:

```
widget.bind(event, handler)
```

Parameters:

- **event:** this parameter is used to define the events so that it can be handled using the handler function. E.g FocusIn, Enter, KeyPress, etc
- **handler:** this parameter is used to define the handler function so that it can describe the event that occurred using the event objects which are called along with the handler function such as having a mouse position using the x and y-axis in pixels, mouse button numbers, etc.

Event	Description
<Button>	A mouse button is pressed with the mouse pointer over the widget. The detail part specifies which button, e.g. The left mouse button is defined by the event <Button-1>, the middle button by <Button-2>, and the rightmost mouse button by <Button-3>. <Button-4> defines the scroll up event on mice with wheel support and and <Button-5> the scroll down. If you press down a mouse button over a widget and keep it pressed, Tkinter will automatically "grab" the mouse pointer. Further mouse events like Motion and Release events will be sent to the current widget, even if the mouse is moved outside the current widget. The current position, relative to the widget, of the mouse pointer is provided in the x and y members of the event object passed to the callback.
<Key>	The user pressed any key. The key is provided in the char member of the event object passed to the callback

The Tkinter bind is defined as a function for **dealing with the functions and methods** of Python that are bind to the events that occur during the program execution such as moving the cursor using a mouse, clicking of mouse buttons, clicking of buttons on the keyboard, etc are **events that are handled using bind function in Tkinter.**

Tkinter is used for designing the web applications or desktop applications where there are different **events created such as navigating from one page to another, clicking on one link and going to another page,** etc are all events and such events can be handled by using a binding function known as bind() which we can bind any functions of Python to such events to handle the events and then these functions, in turn, can be bind with any widgets provided by the Tkinter module.

A lambda function is a small anonymous function.

A lambda function can take any number of arguments, but can only have one expression.

Syntax

lambda arguments : expression

Add 50 to argument y, and return the result:

```
x = lambda y : y + 50
print(x(10))
```

The **grid method is used to structure the parent widgets in table-like structure (as against blocks)** before placing them in the main window

Syntax

widget.grid(grid_options)

The grid_options are :

1. **column**
2. **columnspan**
3. **ipadx, ipady**
4. **padx, pady**
5. **row**
6. **rowspan**
7. **sticky**

The **Place** geometry manager allows you to explicitly set the position and size of a window, either in absolute terms, or relative to another window.

Syntax:

widget.place(relx = 0.5, rely = 0.5, anchor = CENTER)

Note : place() method can be used with grid() method as well as with pack() method.

options −

- **anchor** − N, E, S, W, NE, NW, SE, or SW, compass directions indicating the corners and sides of widget; default is NW (the upper left corner of widget)
- **bordermode** − INSIDE (the default) to indicate that other options refer to the parent's inside (ignoring the parent's border); OUTSIDE otherwise.
- **height, width** − Height and width in pixels.
- **relheight, relwidth** − Height and width as a float between 0.0 and 1.0, as a fraction of the height and width of the parent widget.
- **relx, rely** − Horizontal and vertical offset as a float between 0.0 and 1.0, as a fraction of the height and width of the parent widget.
- **x, y** − Horizontal and vertical offset in pixels.

set() and get() methods are used to set and retrieve the values
The values of these variables can be set using set() method

There are 4 tkinter variables.

BooleanVar()
StringVar()
IntVar()
DoubleVar()

```
'''
import tkinter
from tkinter import *
window=Tk()
window.title("GUI Application in Python Environment")

pradnya=Label(window,text="Hello how are you doing")
#pradnya.pack()
pradnya.pack(side=LEFT,expand=True,padx=10,pady=30)
window.mainloop()

'''
'''
def display():
    print("Hi This is Button widget")
button1=Button(window,text="Click to see the message",
          command=display,bg='red',fg='green',font='arielblack',
          height=10,width=20,bd=10)
button1.pack()
```

```
'''
import tkinter.messagebox
def callback():
    tkinter.messagebox.showinfo("Message box would appear as title","Hi How are YOU!")

def flatc():
    print("Hello ! Enjoying learing GUI features ")

'''
from tkinter import *
import tkinter

relief_attrib=tkinter.Tk()

#relief_attrib.geometry("600x600")

relief_flat=tkinter.Button(relief_attrib,text="FLAT",
              relief=FLAT,bg='green',fg='red',font='arielblack',
                  command=flatc)
```

```
relief_raised=tkinter.Button(relief_attrib,text="RAISED",state=NORMAL,
                relief=RAISED,bg='orange',fg='yellow',cursor='pencil')
relief_sunken=tkinter.Button(relief_attrib,text="SUNKEN",relief=SUNKEN,
                                        command=callback,cursor='tcross',
                      font=('times','28','italic'),fg='blue',bg='red')

relief_groove=tkinter.Button(relief_attrib,text="GROOVE",relief=GROOVE,
                      cursor='hand2',bg='yellow',fg='brown')

relief_ridge=tkinter.Button(relief_attrib,text="RIDGE",relief=RIDGE,
                     cursor='heart',bg='red',fg='green')

#quitb= tkinter.Button(relief_attrib,text="QUIT",command=relief_attrib.destroy)
quitb= tkinter.Button(relief_attrib,text="QUIT",command=quit)
quitb.pack(side=BOTTOM)

relief_flat.pack(side=LEFT,expand=True,padx=10,pady=30)

relief_raised.pack(side=RIGHT,expand=True,padx=10,pady=30)

relief_sunken.pack(side=TOP,expand=True,padx=10,pady=30)

relief_groove.pack(side=BOTTOM,expand=True,padx=10,pady=30)
```

```
relief_ridge.pack(side=LEFT,expand=True,padx=10,pady=30)
```

```
from tkinter import *

def opted():
   selection = "Selected category under OPTION NO.:  ---- " + str(var.get())
 label.config(text = selection)

mainwindow = Tk()
var = IntVar()
option1 = Radiobutton(mainwindow, text="Sports: 01", variable=var, value=1,
                command=opted)
option1.pack( anchor = N )

option2 = Radiobutton(mainwindow, text="Entertainment: 02", variable=var, value=2,
                command=opted)
option2.pack( anchor = E )

option3 = Radiobutton(mainwindow, text="Travelling :03", variable=var, value=3,
                command=opted)
option3.pack( anchor = W)

option4 = Radiobutton(mainwindow, text="Electronics :04", variable=var, value=4,
                command=opted)
```

```
option4.pack( anchor = S)
label = Label(mainwindow)
label.pack()
mainwindow.mainloop()
'''


'''
from tkinter import *



root_window = Tk()

root_window.geometry("400x400")

button_obj = Button(root_window,text = "Computer Science")

button_obj.pack()

root_window.mainloop()

'''
```

```
from tkinter import *

parent_window = Tk()
parent_window.geometry("400x300")

label_obj = Label(parent_window, text ='Computer Science Learners ',
              fg="Red",font = "100")
label_obj.pack()
#No shared variable in contrast to radio button
Checkbuttonvar1 = IntVar()
Checkbuttonvar2 = IntVar()
Checkbuttonvar3 = IntVar()
chk_obj1 = Checkbutton(parent_window, text = "Entertainment", font="50",
                                    variable = Checkbuttonvar1,
                                    onvalue = 1,
                                    offvalue = 0,
                                    height = 20,
                                    width = 30)
```

```
chk_obj2 = Checkbutton(parent_window, text = "Sports",
                                                              variable = Checkbuttonvar2,
                                                              onvalue = 1,
                                                              offvalue = 0,
                                                              height = 2,
                                                              width = 10)
chk_obj3 = Checkbutton(parent_window, text = "Hobbies",
                                                              variable = Checkbuttonvar3,
                                                              onvalue = 1,
                                                              offvalue = 0,
                                                              height = 20,
                                                              width = 30)
def var_states():
  print("Entertainment: %d,\nSports: %d,\nHobbies:  %d" % (Checkbuttonvar1.get(),
                              Checkbuttonvar2.get(),
                              Checkbuttonvar3.get()))

chk_obj1.pack()
chk_obj2.pack()
chk_obj3.pack()
Button(parent_window, text='Quit', command=parent_window.destroy).pack(side=RIGHT)
Button(parent_window, text='Show', command=var_states).pack()

parent_window.mainloop()
```

Unit 3

Database connectivity in Python

Installing mysql connector, accessing connector module, using connect, cursor, execute & close functions, reading single & multiple results of query execution, executing different types of statements, executing transactions, understanding exceptions in database connectivity.

Network connectivity

Socket module, creating server-client programs, sending email, reading from URL