UNIVERSITY of TASMANIA

KIT100 PROGRAMMING PREPARATION

Lecture Eleven:

GUI Components part 2

Lecture Objectives

- Canvas Widget
- The Geometry Managers
- Displaying Images
- Menus
- Working through GUI examples

Introduction

- Last week
  - we have covered the basic elements of creating GUIs in Python.

- Object-Oriented Programming is the preferred method for many programmers.
  - We are using “classes” in GUI program and not procedural coding.

The Canvas widget

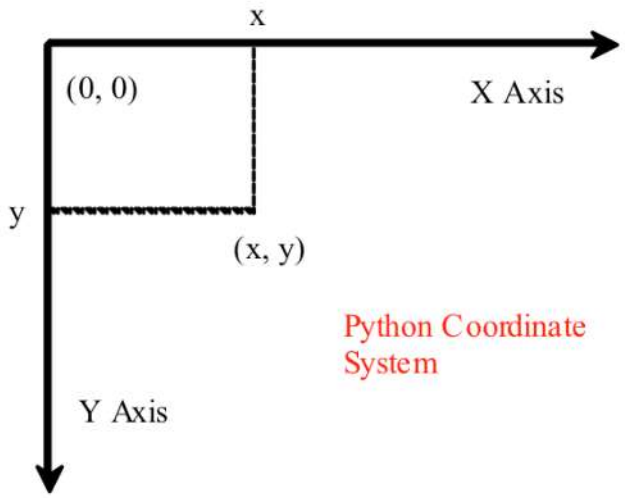
- We can use the Canvas widget for displaying shapes.
  - The Canvas is a “rectangular area” intended for drawing pictures or other complex layouts.
  - Such as draw graphs and plots, create graphics editors, and implement custom widgets. So you can place graphics, texts, widgets, or frames on a Canvas.

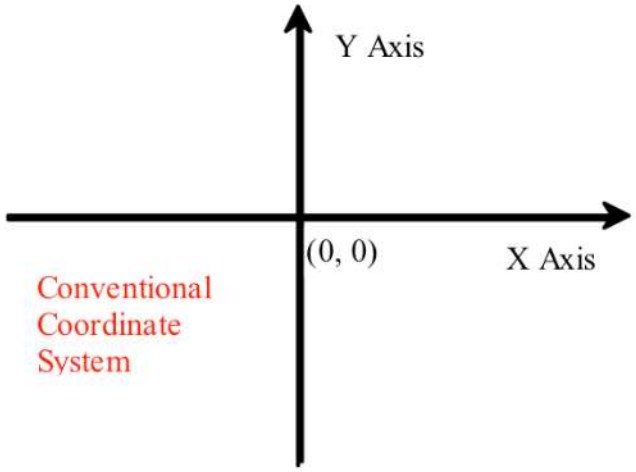
- The `Canvas` widget has several methods, some of which are:

```
create_rectangle
create_oval
create_arc
create_polygon
create_line
```

Co-ordinate System

- To draw graphics, we need to tell the widget where to draw.
- Each widget has its own co-ordinate system with the origin (0,0) at the upper-left corner.
  - The x co-ordinate increases to the right, and the y-coordinate increases downwards.
- The Tkinter coordinate system differs from the conventional coordinate system.
- We need to remember this when entering coordinates for drawing shapes.





create_rectangle

Create a rectangle:

- It is bounded by the box given by the x,y coordinates of the top left-hand corner and bottom right-hand corner

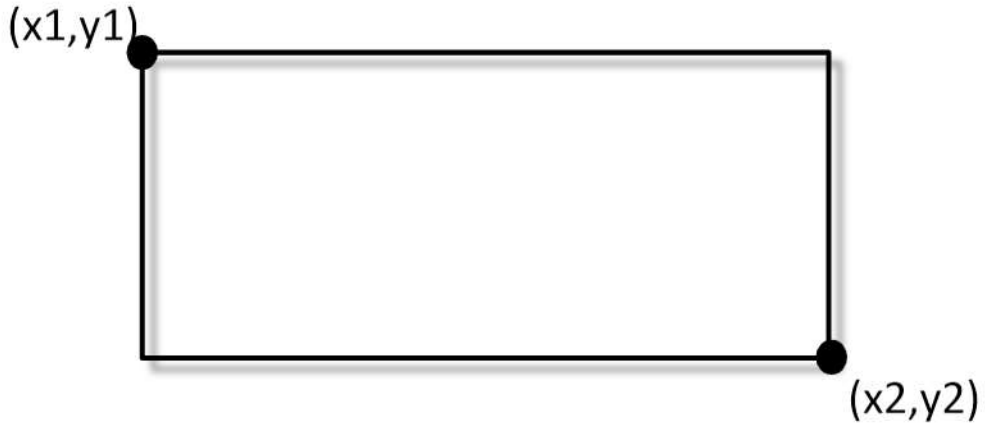
Syntax:

```
create_rectangle(bbox, options)# bbox: the boundary of the box.
```

Example:

```
self.canvas.create_rectangle(10,10,190,90, tags = "rect")
```

create_rectangle




```
canvas.create_rectangle(x1, y1, x2,  y2, tags = "rect")
canvas.create_rectangle(10, 10, 190, 90, tags = "rect")
```

create_oval

Create an oval:

- it is bounded by the box given by the x,y coordinates of the top left-hand corner and bottom right-hand corner.

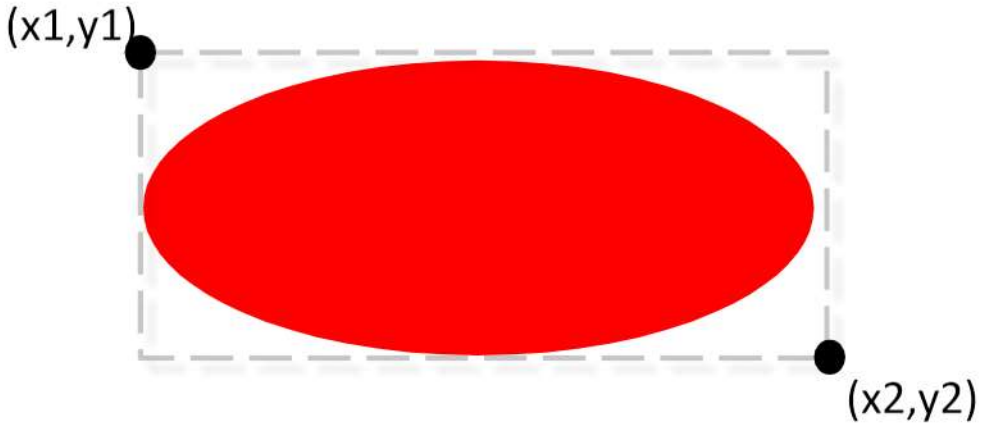
Syntax:

```
create_oval(bbox, options)  # bbox: the boundary of the box.
```

Example:

```
canvas.create_oval(10,10,190,90,fill = "red", tags = "oval")
```

create_oval




```
canvas.create_oval(10, 10, 190, 90, fill="red", tags = "oval")
```

create_line

Create a line:

- It is bounded by the box given by the x,y coordinates of the top left-hand corner and bottom right-hand corner.

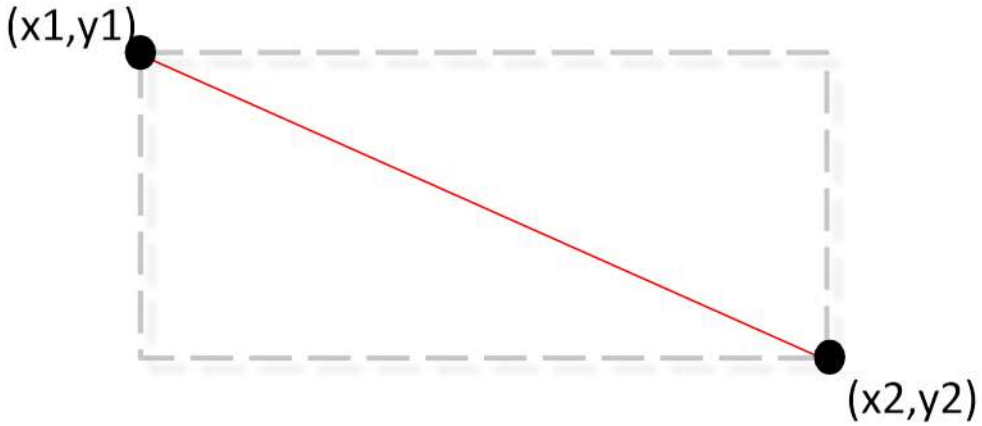
Syntax:

```
create_line(coords, options)
```

Example:

```
canvas.create_line(10,10,190,90,fill = “red”, tags = "line")
```

create_line




```
canvas.create_line(10, 10, 190, 90, fill="red" tags = "line")
```

create_arc

Create an arc:

- It is bounded by the box given by the x,y coordinates of the top left-hand corner and bottom right-hand corner.

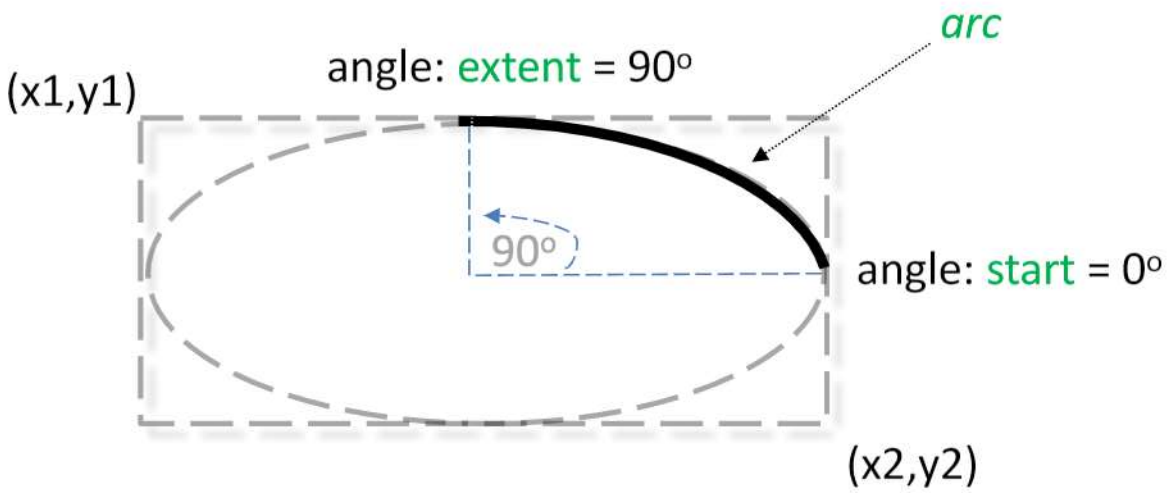
Syntax:

```
create_arc(bbox, options)
```

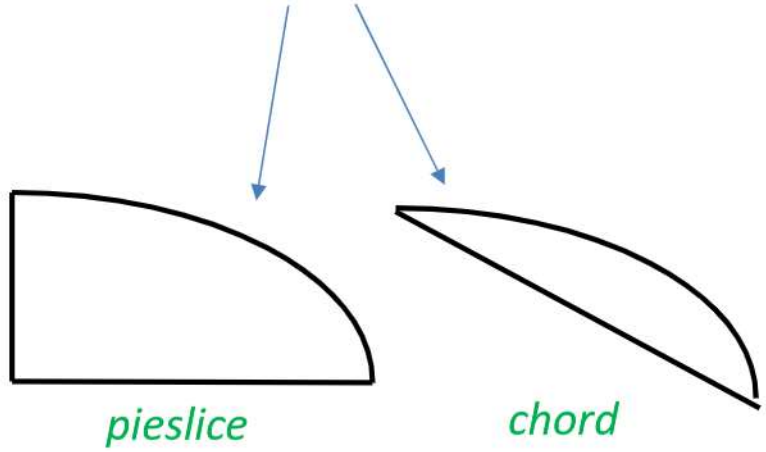
Example:

```
canvas.create_arc(10,10,190,90,start=0,extent=90,width=2,
                  style="arc", tags="arc")

style: "arc", "pieslice", "chord"
                                    # start/extent: start/extent angle;
                                          # width: line width;
```

create_arc




```
canvas.create_arc(10,10,190,90,
                  start=0,extent=90,width=2,style="arc", tags = "arc")
```



create_polygon

Create a polygon:

- It is bounded by the x,y coordinates of each vertex (minimum of three coordinates).

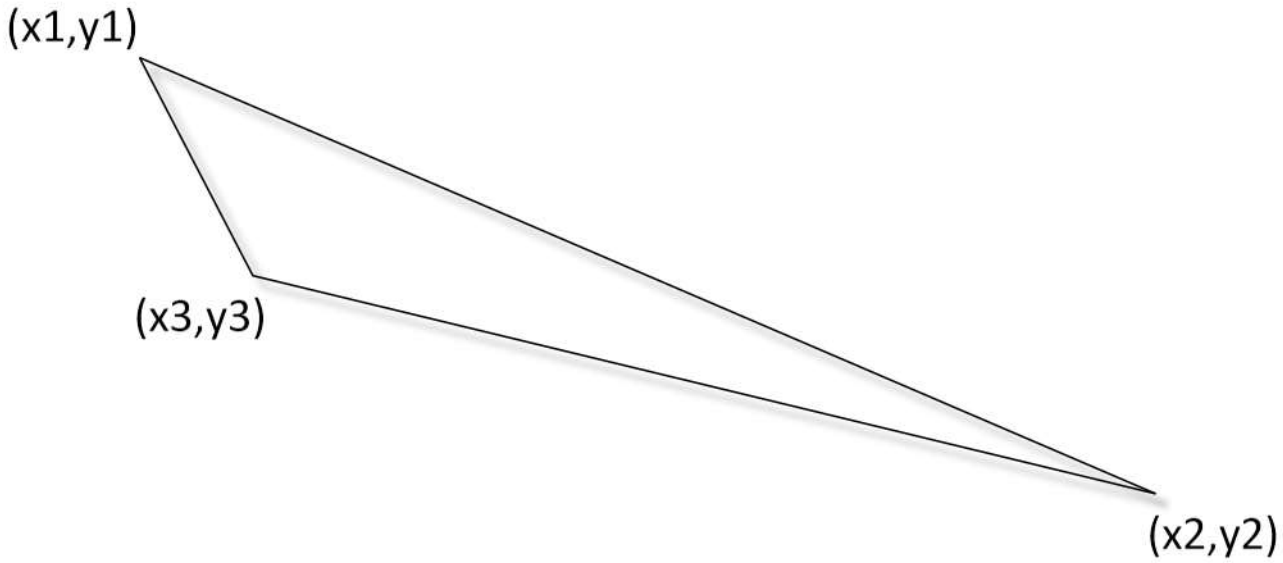
Syntax:

```
create_polygon(coords, options)
```

Example:

```
                          x1,y1 x2,y2  x3,y3
canvas.create_polygon(10,10,190,90,30,50,tags="polygon")
```

create_polygon




```
create_polygon(x1,y1,x2,y2,x3,y3)
```

```
canvas.create_polygon(10,10,190,90,30,50,tags="polygon")
```

Using the Canvas – CanvasDemo.py


```
from tkinter import * # Import Tkinter

class CanvasDemo:

    def __init__(self):

        window = Tk() # Create a window
        window.title("Canvas Demo") # Set title

        # Place self.canvas in the window
        self.canvas = Canvas(window, width = 200, height = 100, bg = "white")
        self.canvas.pack()

        # Place buttons in frame
        frame = Frame(window)
        frame.pack()

        btRectangle = Button(frame,text = "Rectangle",command = self.displayRect)

        btOval = Button(frame,text = "Oval", command = self.displayOval)

        btArc = Button(frame,text = "Arc",command = self.displayArc)

        btPolygon = Button(frame,text = "Polygon",command = self.displayPolygon)

        btLine = Button(frame,text = "Line",command = self.displayLine)

        btString = Button(frame,text = "String",command = self.displayString)

        btClear = Button(frame,text = "Clear",command = self.clearCanvas)
|
```

Using the Canvas – CanvasDemo.py

Continue from the previous slide,

```
    btRectangle.grid(row = 1, column = 1)
    btOval.grid(row = 1, column = 2)
    btArc.grid(row = 1, column = 3)
    btPolygon.grid(row = 1, column = 4)
    btLine.grid(row = 1, column = 5)
    btString.grid(row = 1, column = 6)
    btClear.grid(row = 1, column = 7)

    window.mainloop() # Create an event loop

# Display a rectangle
def displayRect(self):
    self.canvas.create_rectangle(10, 10, 190, 90, tags = "rect")

# Display an oval
def displayOval(self):
    self.canvas.create_oval(10, 10, 190, 90, fill = "red", tags = "oval")

# Display an arc
def displayArc(self):
    self.canvas.create_arc(10, 10, 190, 90,
                           start = 0, extent = 90, width = 1, style="arc", tags = "arc")
```

The active fill argument makes the shape change colour when you move the mouse over it.

The width argument can be used to specify the pen size in pixels for drawing the shapes.

Using the Canvas – CanvasDemo.py

Continue from the previous slide,

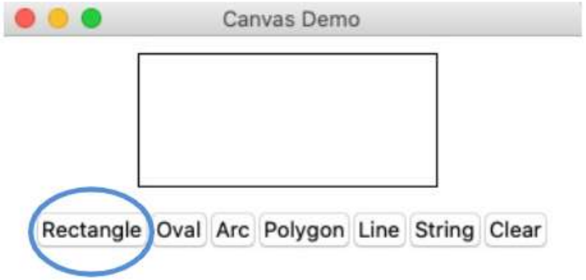
```
    # Display a polygon
    def displayPolygon(self):
        self.canvas.create_polygon(10, 10, 190, 90, 30, 50,
                                   outline="black", fill="", tags = "polygon")

    # Display a line
    def displayLine(self):
        self.canvas.create_line(10, 10, 190, 90, fill = "red", tags = "line")

    # Display a string
    def displayString(self):
        self.canvas.create_text(60, 40, text = "Hi, I am a string",
                                font = "Times 10 bold underline", tags = "string")

    # Clear drawings
    def clearCanvas(self):
        self.canvas.delete("rect", "oval", "arc", "polygon", "line", "string")

CanvasDemo() # Create GUI
```

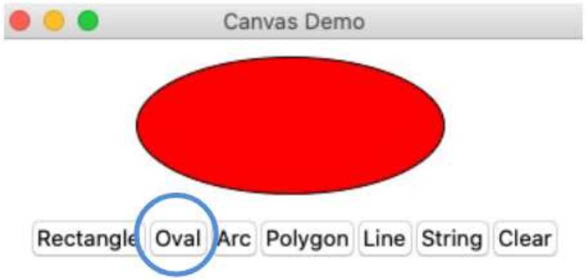
All the drawing tags are used in the delete method for clearing the drawing from the canvas.

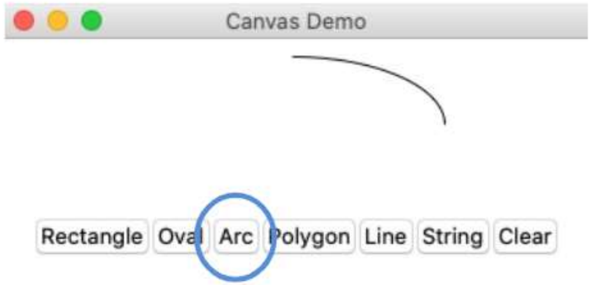
The create_text method is used to draw a text string.
Note that the horizontal and vertical center of the text is displayed at (x,y) for create_text(x, y, text).

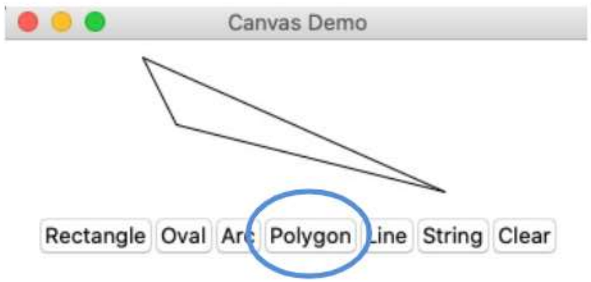
Output

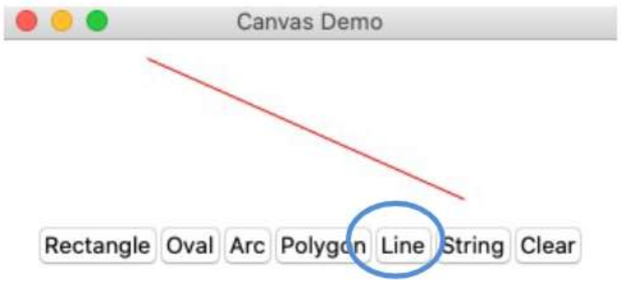
Default

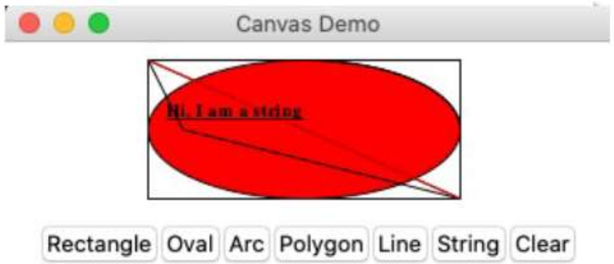












Show All

Clear All

The Geometry Managers

- Last week
  - Tkinter uses a geometry manager (pack/grid) to place widgets inside a container (window).

- Let's review each manager for layout purpose.
  - The grid manager
  - The pack manager
  - The place manager

The Grid Manager

- The grid manager places widgets into the cells of an invisible grid in a container.
  - We can place a widget in a specified row and column.
  - We can also use the rowspan and columnspan parameters to place a widget in multiple rows and columns.

row 1, column 1	row 1, column 2	row 1, column 3	row 1, column 4	
row 2, column 1	row 2, column 2	row 2, column 3	row 2, column 4	Etc..
row 3, column 1	row 3, column 2	row 3, column 3	row 3, column 4	
		Etc..		
				21

Using the Grid Manager – GridManagerDemo.py

```
from tkinter import * # Import tkinter

class GridManagerDemo:

    window = Tk() # Create a window
    window.title("Grid Manager Demo") # Set title

    message = Message(window, text =
                      "This Message widget occupies three rows and two columns")

    message.grid(row = 1, column = 1, rowspan = 3, columnspan = 2)

    Label(window, text = "First Name:").grid(row = 1, column = 3)

    Entry(window).grid(row = 1, column = 4, padx = 5, pady = 5)

    Label(window, text = "Last Name:").grid(row = 2, column = 3)

    Entry(window).grid(row = 2, column = 4, padx = 5, pady = 5)

    Button(window, text = "Get Name").grid(row = 3, padx = 2, pady = 2,
                                           column = 4, sticky = E)

    window.mainloop() # Create an event loop

GridManagerDemo() # Create GUI
```

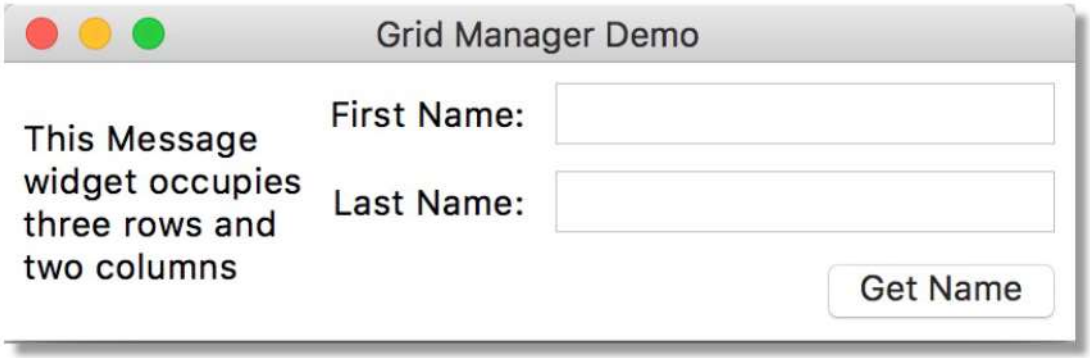
The Message widget is placed in row 1 and column 1 and it expands to three rows and two columns.

The padx and pady options pad the optional horizontal and vertical space in a cell.

'sticky = E' option to stick to the east in the cell so that it is right aligned with the Entry widgets in the same column.

The sticky option can be any combination of the named constants S, N, E and W, or NW, NE, SW, and SE.

Output


Grid Manager Demo
This Message widget occupies three rows and two columns
First Name:
Last Name:
Get Name

The Pack Manager

- The pack manager can place widgets
  - on top of each other (pack(side =“top”))
    - this is the default mode.
  - or place them side by side (pack(side =“left”)).

- Last week
  - we can use the pack manager to organize the layout of our widgets.
  - we can also use the fill option to make a widget fill its entire container (window).
    - ➢ The fill option uses named constants X, Y, or BOTH to fill horizontally, vertically, or both ways.

Using the Pack Manager - PackDemo.py

Here we have code for three labels.

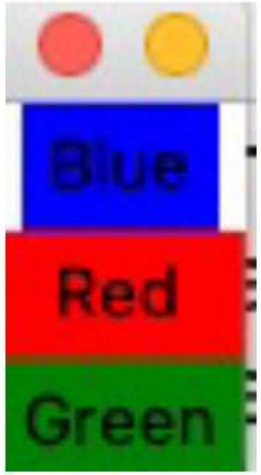
- These three labels are packed on top of each other (the default mode).

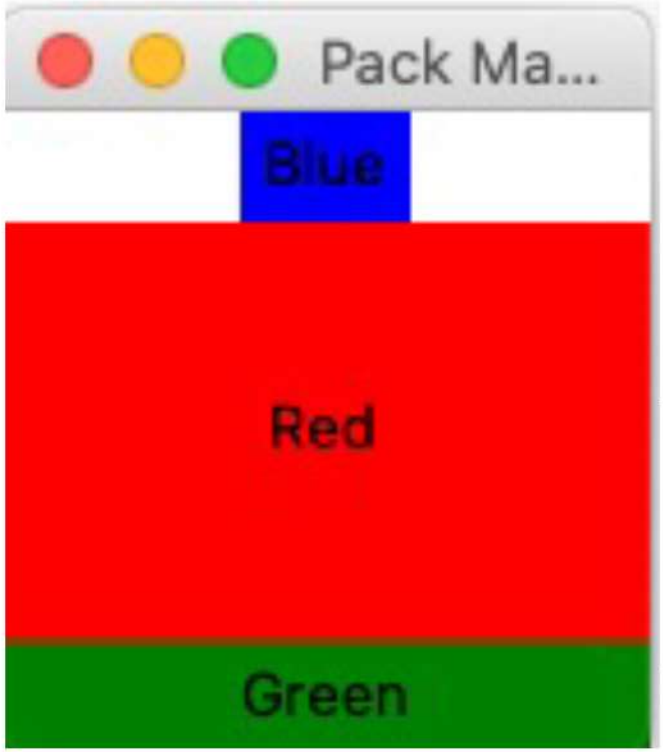
```
from tkinter import * # Import tkinter

class PackManagerDemo:

    def __init__(self):

        window = Tk() # Create a window
        window.title("Pack Manager Demo 1") # Set title

        Label(window, text = "Blue", bg = "blue").pack()
        Label(window, text = "Red", bg = "red").pack(fill=BOTH, expand=1)
        Label(window, text = "Green", bg = "green").pack(fill = X)

        window.mainloop() # Create an event loop

PackManagerDemo() # Create GUI
```

The fill option uses named constants X, Y, or BOTH to fill horizontally, vertically, or both ways.

The expand option tells the pack manager to assign additional space to the widget box. If the parent widget is larger than necessary to hold all the packed widgets, any extra space is distributed among the widgets whose expand option is set to a nonzero value.

Output





BOTH:
Horizontally and
vertically fill in.

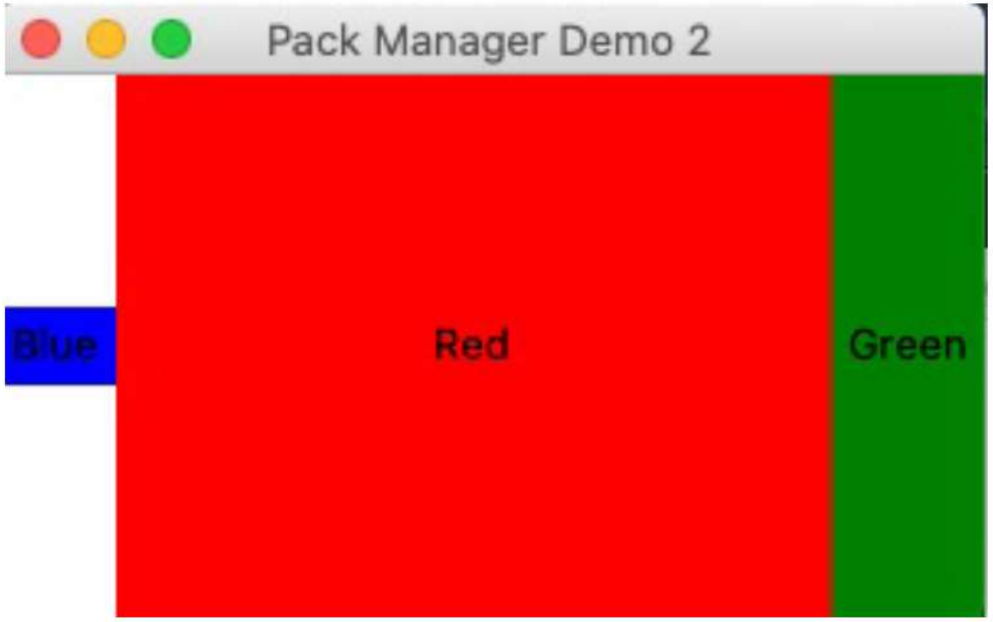
X:
Horizontally fill in.

Pack Manager –PackWithSide.py

Another example,
Here with have the pack option without the default option of 'TOP' - use side

```
from tkinter import * # Import tkinter

class PackManagerDemoWithSide:

    window = Tk() # Create a window
    window.title("Pack Manager Demo 2") # Set title

    Label(window, text = "Blue", bg = "blue").pack(side = LEFT)
    Label(window, text = "Red", bg = "red").pack(side = LEFT, fill = BOTH, expand = 1)
    Label(window, text = "Green", bg = "green").pack(side = LEFT, fill = Y)

    window.mainloop() # Create an event loop

PackManagerDemoWithSide() # Create GUI
```

Output



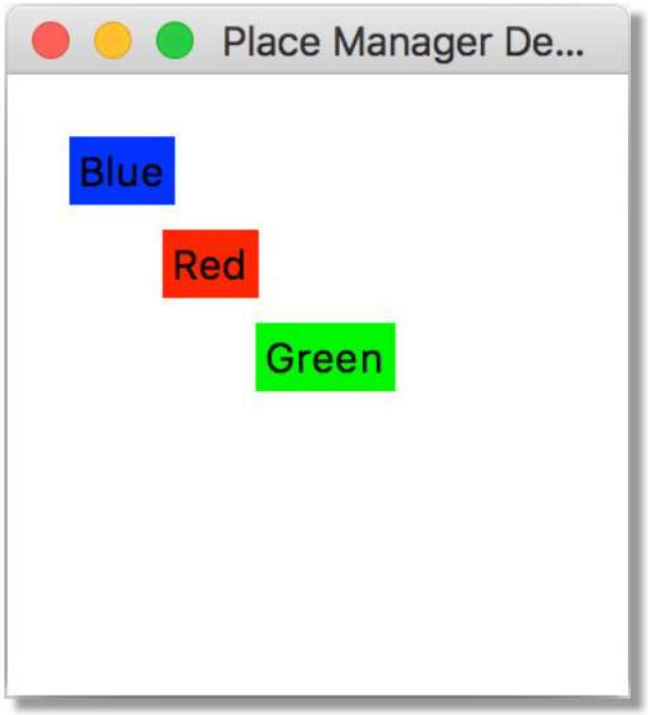
The Place Manager – PlaceManager.py

- The place manager* places widgets in absolute positions.

```
from tkinter import * # Import tkinter

class PlaceManagerDemo:

    def __init__(self):

        window = Tk() # Create a window
        window.title("Place Manager Demo") # Set title

        Label(window, text = "Blue", bg = "blue").place(x = 20, y = 20)
        Label(window, text = "Red", bg = "red").place(x = 50, y = 50)
        Label(window, text = "Green", bg = "green").place(x = 80, y = 80)

        window.mainloop() # Create an event loop

PlaceManagerDemo() # Create GUI
```

* The place manager is not compatible with all computers. For this reason we tend to avoid using the place manager.

Output



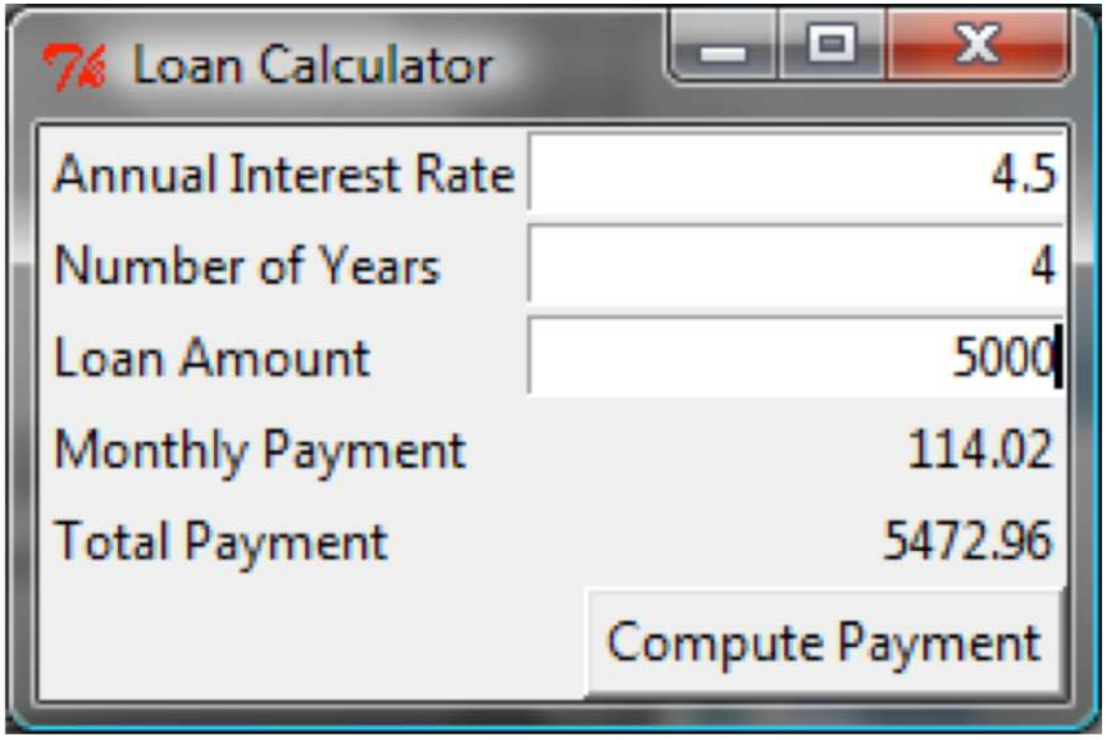
Knowing where to place it all!

- In summary,
  - developing a GUI application involves designing the user interface and then writing the code to process the events.

- Remember before you can code anything you need to plan it.
  - 1. Design the interface
  - 2. Process the event (by writing the code)

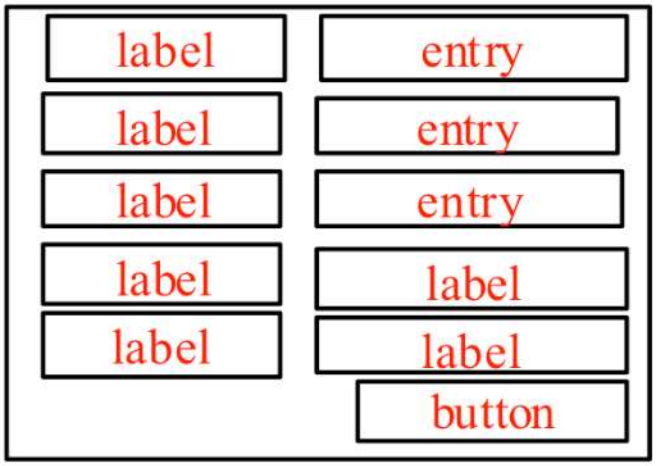
Designing the Layout

Start to think about the interface layout…

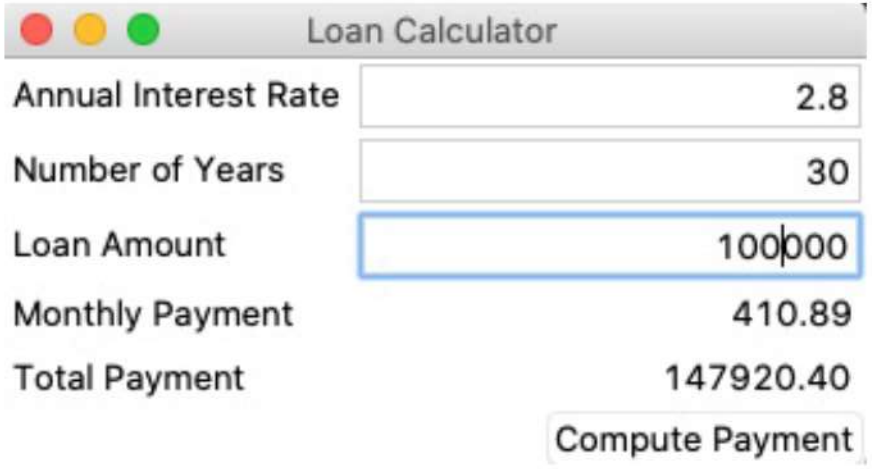


The sketch for the calculator

Sketch



Output



LoanCalculator_p33.py

Displaying Images

- We can add an image to a label, button, check button, or radio button.
  - The image file must be in GIF format.
  - We need to keep the image files in the same file as our Python applications.

- To create an image we can use the `PhotoImage`
  - Creates a reference to a picture (but does not display it yet)

- Syntax:

```
photo = PhotoImage(file = imagefilename)
```

Displaying Images – PhotoImage.py

PhotoImage_p35.py - /Users/czh513/Desktop/KIT001/Teaching in 2020/1 Lecture/week11/examples/

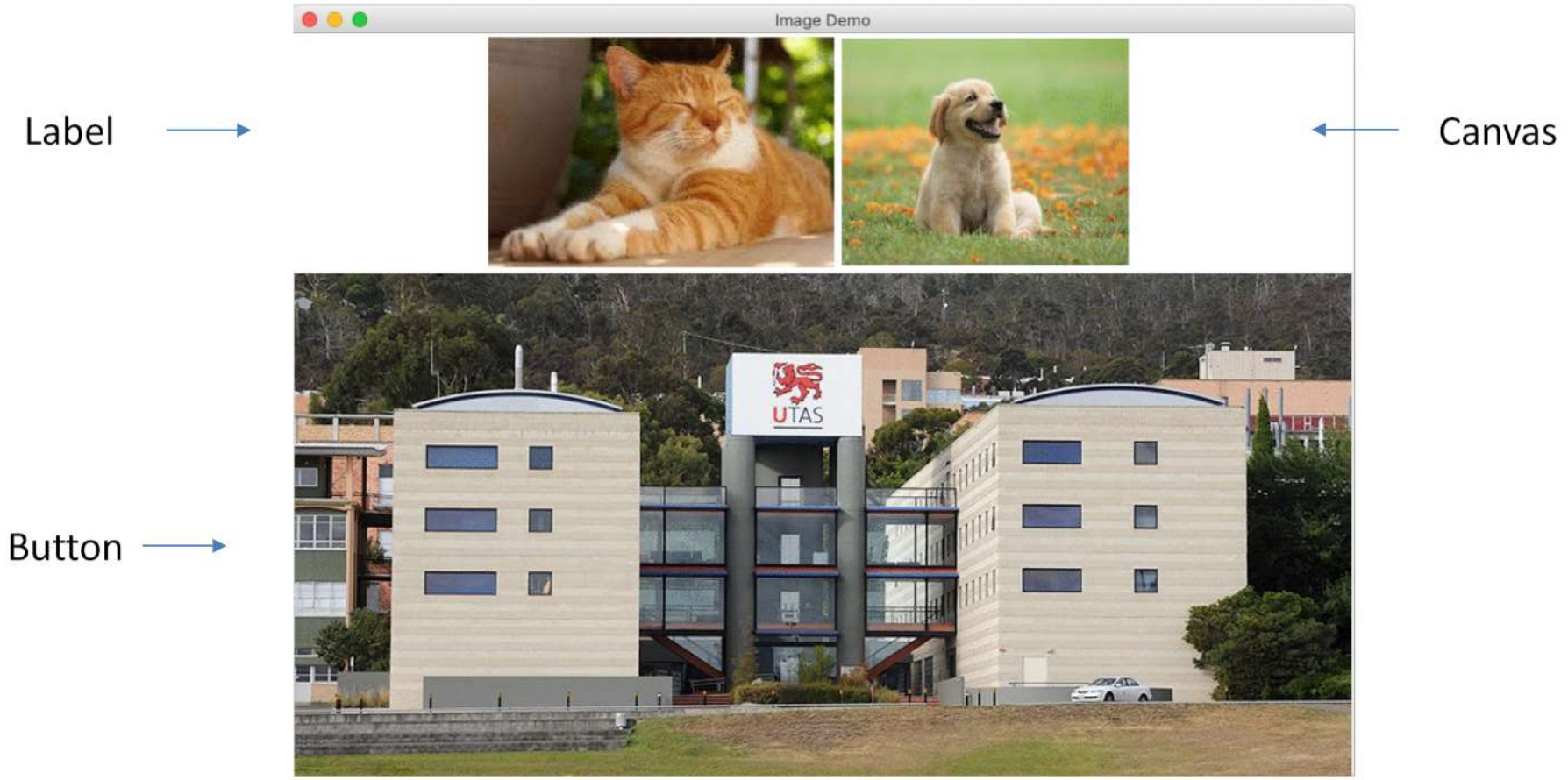
```
from tkinter import *

class ImageDemo:
    def __init__(self):

        window = Tk()
        window.title("Image Demo")

        # Create PhotoImage objects
        catImage = PhotoImage(file = "images/cat.gif")
        dogImage = PhotoImage(file = "images/dog.gif")
        utasImage= PhotoImage(file = "images/utas.gif")

        #frame1 to contain label and canvas
        frame1 = Frame(window)
        frame1.pack()

        Label(frame1,image = catImage).pack(side = LEFT)

        canvas = Canvas(frame1)
        canvas.create_image(90,90, image = dogImage)
        canvas["width"] = 250
        canvas["height"] = 180
        canvas.pack(side = LEFT)
```

create_image(x0,y0, options ...) is used to draw an image from (x0,y0) on a canvas.

Displaying Images – PhotoImage.py

```
        #frame2 to contain button
        frame2 = Frame(window)
        frame2.pack()

        Button(frame2,image = utasImage).pack(side = LEFT)

        window.mainloop()

ImageDemo() #Create the GUI
```

Output



Menus

- Menus make selection easier and are widely used in windows of GUI's.

- We can use Tkinter to create menus,
  - pop up menus and toolbars.

- We can use the Menu class to create a menu bar and a menu, and use the add_command method to add items to the menu.

- Layout of Menus will differ between the different operating systems – so my example from a Mac will look different from when you work through on a PC.

Menus Demo – SimpleMenu.py

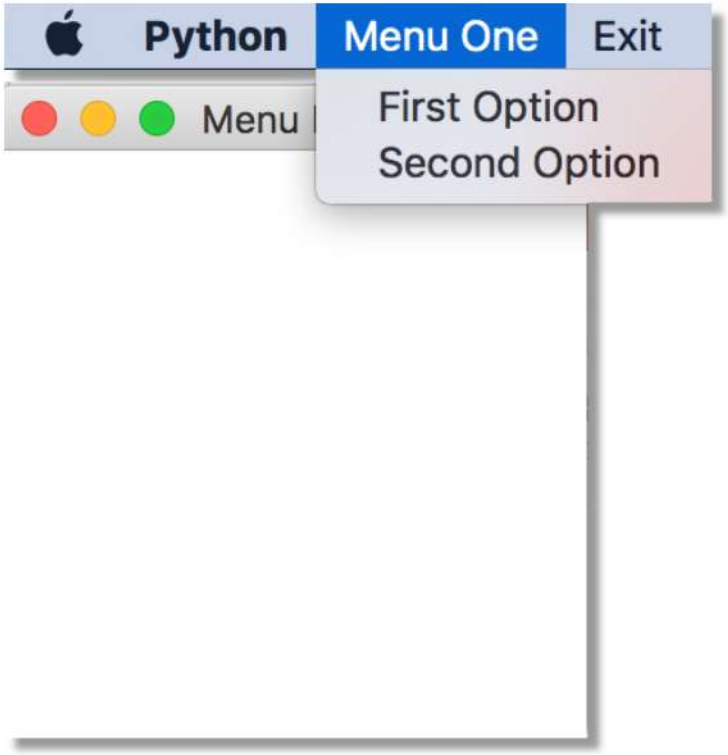
```
from tkinter import *

class MenuDemo:
    def __init__(self):

        window = Tk()
        window.title("Menu Demo")

        # Create a menu bar
        menubar = Menu(window)
        window.config(menu = menubar) # Display the menu bar

        # Create a pull-down menu, and add it to the menu bar
        menuOne = Menu(menubar)

        menubar.add_cascade(label = "Menu One", menu = menuOne)
        menuOne.add_command(label = "First Option")
        menuOne.add_command(label = "Second Option")

        exitMenu = Menu(menubar)

        menubar.add_cascade(label = "Exit", menu = exitMenu)
        exitMenu.add_command(label = "Quit")

        mainloop()

MenuDemo() # Create GUI
```

To display the menu, we use the config method to add the menu bar to the container

To create a menu inside the menu bar

To add items to the menu

Output



GUI Examples

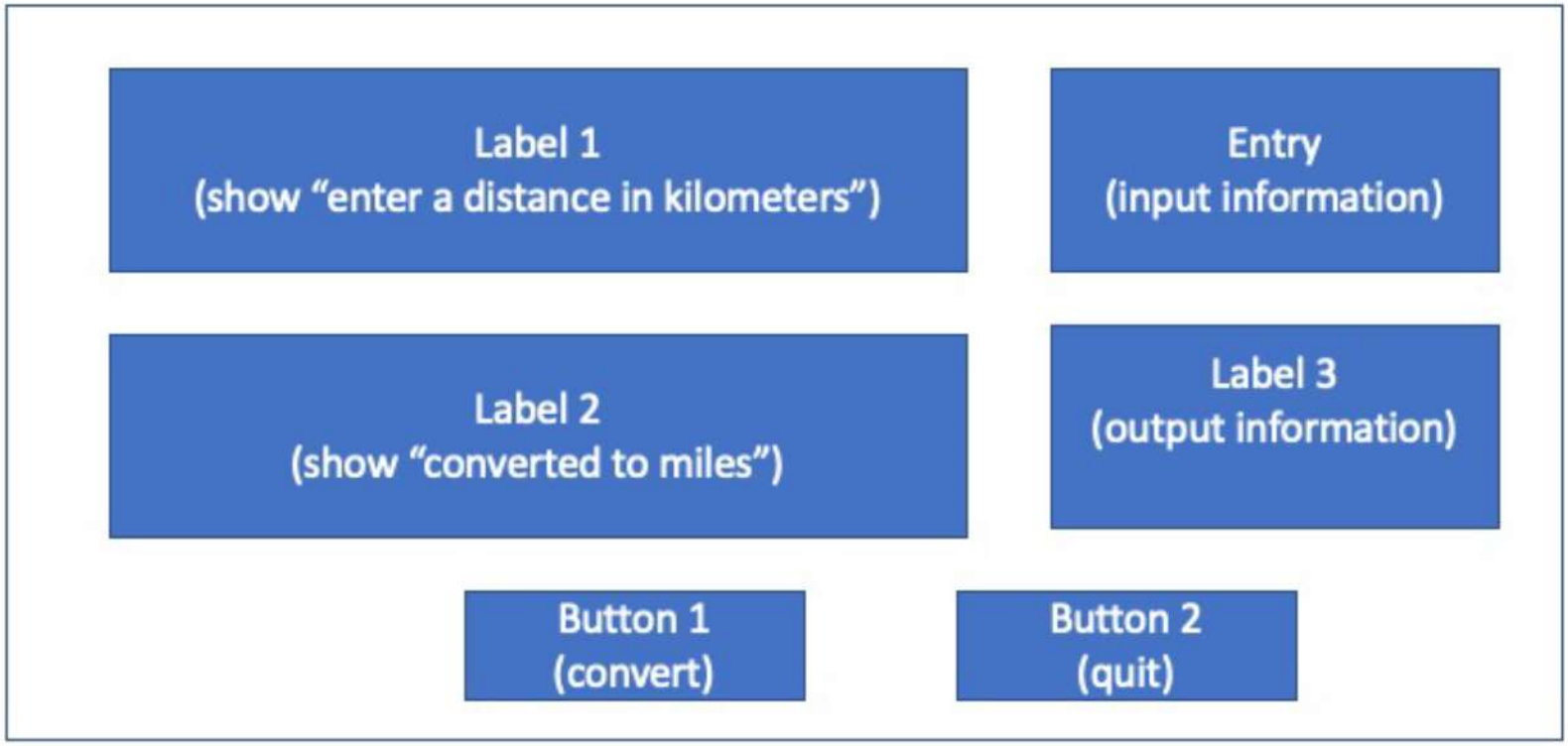
- So now we have worked through the basic components of GUI application with Python.

- Lets go through GUI examples.
  - Let's say we want to create a GUI for converting Kilometres to Miles.

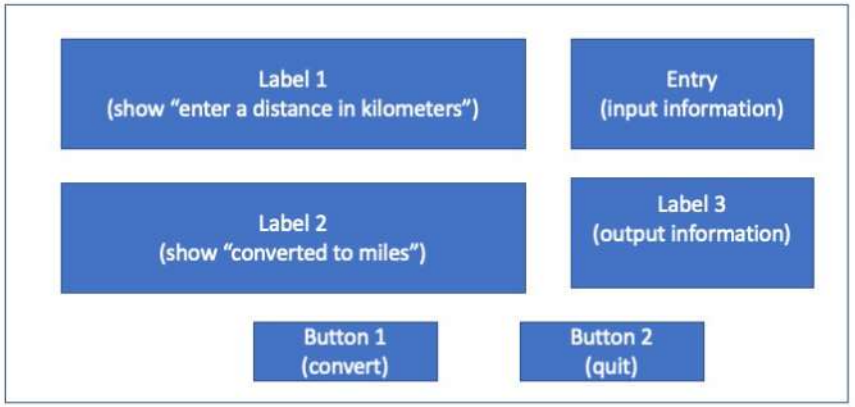
Converting Kilometres to Miles

- Create a GUI for converting Kilometres to Miles.
  - What will our GUI look like?

Sketch



Converting Kilometres to Miles



What we can see?

1. A label widget with the text: Enter a distance in kilometres:
2. We have an Entry widget next to that.
3. We then have another label with the text "Converted to miles:"
4. Another label next to the above – this label should update with our programs output.
5. We have two button widgets, Convert and Quit.
6. We can see we have three frames in this GUI – a top, middle and a bottom frame.

The Code – ExampleGUI.py


```
from tkinter import *

class KiloConverterGUI:

    def __init__(self):

        self.main_window = Tk()

        # some frames for the layout
        self.top_frame = Frame()
        self.mid_frame = Frame()
        self.bottom_frame = Frame()

        self.prompt_label = Label(self.top_frame,text="Enter a distance in kilometers: ")
        self.prompt_label.pack(side='left')

        self.kilo_entry = Entry(self.top_frame,width=10)
        self.kilo_entry.pack(side='left')

        self.descr_label = Label(self.mid_frame,text="Converted to miles: ")
        self.descr_label.pack(side='left')

        self.value = StringVar() #Tk function for String values in GUI
        self.miles_label = Label(self.mid_frame,textvariable=self.value)
        self.miles_label.pack(side='left')
```

The Code – ExampleGUI.py


```
        # action buttons
        self.calc_button = Button(self.bottom_frame,text='Convert',
                                  command=self.convert)
        self.calc_button.pack(side='left')

        self.quit_button = Button(self.bottom_frame,text='Quit',
                                  command=self.main_window.destroy)
        self.quit_button.pack(side='left')

        # show the frames
        self.top_frame.pack()
        self.mid_frame.pack()
        self.bottom_frame.pack()

        mainloop()

    def convert(self):

        kilo = float(self.kilo_entry.get())

        miles = kilo * 0.6214

        self.value.set("%.2f"%miles) #value is StringVar type, which has set method

KiloConverterGUI()
```

First Part of the Code: ExampleGUI.py

First Part of the Code,

First we need to create a main window.

Then we need to create three frames to group widgets.

```
from tkinter import *

class KiloConverterGUI:

    def __init__(self):

        self.main_window = Tk()

        # some frames for the layout
        self.top_frame = Frame()
        self.mid_frame = Frame()
        self.bottom_frame = Frame()
```

Second Part of the Code : ExampleGUI.py

Second part of the code,

Now we need to create the widgets for the top frame and call their pack method.

```
self.prompt_label = Label(self.top_frame,text="Enter a distance in kilometers: ")
self.prompt_label.pack(side='left')

self.kilo_entry = Entry(self.top_frame,width=10)
self.kilo_entry.pack(side='left')
```

Third Part of the Code: ExampleGUI.py

Third Part of the Code,
Now we create the widgets for the middle frame.

```
self.descr_label = Label(self.mid_frame,text="Converted to miles: ")
self.descr_label.pack(side='left')

self.value = StringVar() #Tk function for String values in GUI
self.miles_label = Label(self.mid_frame,textvariable=self.value)
self.miles_label.pack(side='left')
```

Note,
We create a StringVar object to associate with an output label, and assign it to the value variable.
We then create a label and associate it with the StringVar object, we named miles_label.
Any value stored in the StringVar object with automatically be displayed in the label.

Forth Part of the Code: ExampleGUI.py

Forth Part of the Code,

Now we create the Button widgets and pack them.

We also pack the Frame objects we created in the first part of the code.

```
# action buttons
self.calc_button = Button(self.bottom_frame,text='Convert',
                          command=self.convert)
self.calc_button.pack(side='left')

self.quit_button = Button(self.bottom_frame,text='Quit',
                          command=self.main_window.destroy)
self.quit_button.pack(side='left')

# show the frames
self.top_frame.pack()
self.mid_frame.pack()
self.bottom_frame.pack()

mainloop()
```

Fifth Part of the Code: ExampleGUI.py

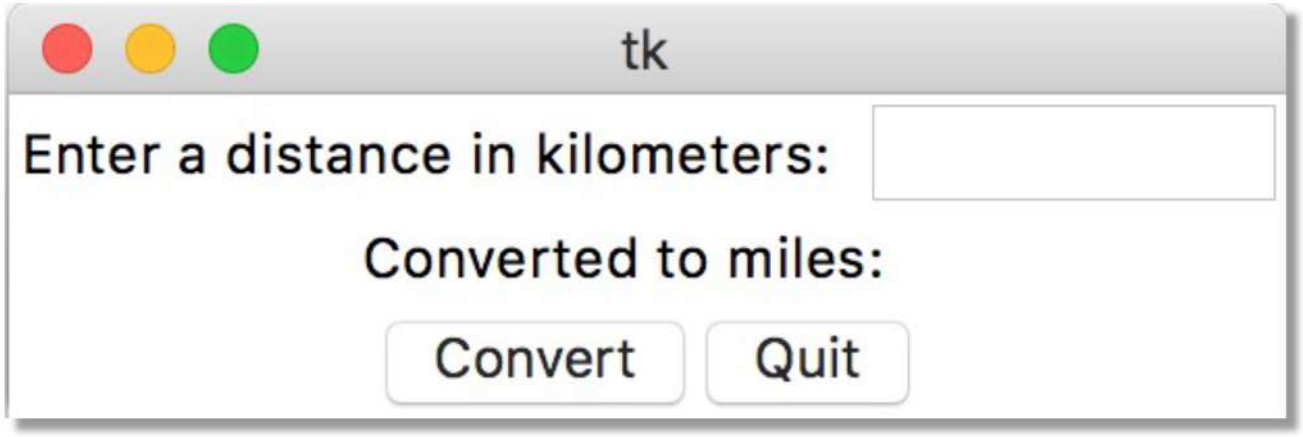
Fifth Part of the Code,

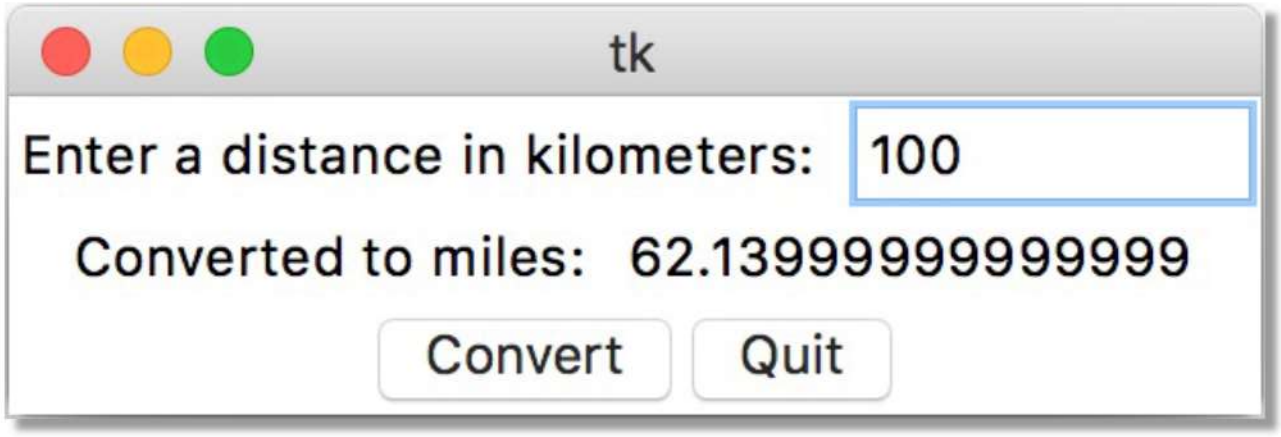
The convert method is the Convert buttons callback function.

```
    def convert(self):
        kilo = float(self.kilo_entry.get())
        miles = kilo * 0.6214
        self.value.set("%.2f"%miles) #value is StringVar type, which has set method
KiloConverterGUI()
```

Note,
So we call the kilo_entry widget's get method to retrieve the data that has been typed into the widget.
Then we call the StringVar obejcts set method (remember the StringVar object has been assigned to the value variable) passing miles as an argument.

Output


tk
Enter a distance in kilometers:
Converted to miles:
Convert
Quit


tk
Enter a distance in kilometers:
100
Converted to miles:
62.13999999999999
Convert
Quit

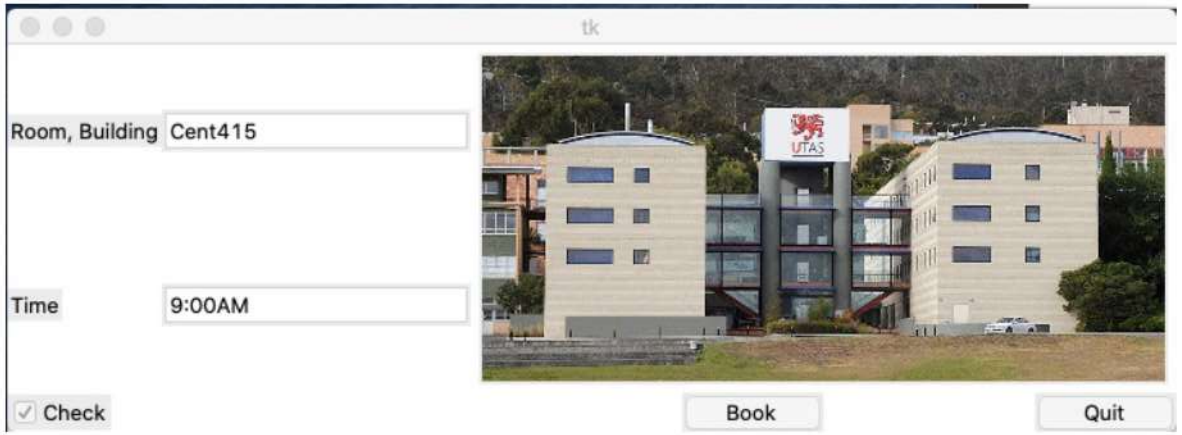
An additional example

UTAS room booking system

Sketch

<table>
<tr><td><label 1></td><td><entry 2></td><td rowspan="2" colspan="2"><image></td></tr>
<tr><td><label 1></td><td><entry 2></td></tr>
<tr><td colspan="2"><checkbutton></td><td><button 1></td><td><button 2></td></tr>
</table>

Output



BookRoom_p52.py

Questions?