UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

UNIDAD DIDÁCTICA V

DIPLOMATURA EN PYTHON

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Módulo II – Nivel Inicial

Unidad V – GUI - Tkinter.

GUI

Label

Tkinter cuenta con varias estructuras preestablecidas, y existe una en particular llamada “Frame” que nos permite organizar el diseño de nuestra aplicación, y es fundamental a la hora de maquetar, ya que cumple varias funciones:

1.- Nos permite agrupar aplicaciones de tkinter para crear diseños complejos.

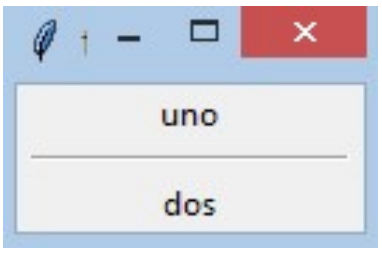
2.- Los podemos utilizar como separadores de secciones si les damos dimensiones y un contenido vacío.

3.- Los podemos utilizar como basé en la creación de aplicaciones.

Su implementación es muy simple, en el siguiente ejemplo lo utilizamos para separar dos textos, de forma similar a como lo haríamos en un documento html mediante el uso de la etiqueta <hr/>. Notar que agregamos explícitamente el “Frame” en la ventana pasando la ventana como primer parámetro. Esta es la forma de indicar que un componente cualquiera se encuentra dentro de otro.

```
from tkinter import *

master = Tk()
Label(text="uno").pack()
separador = Frame(master, height=2, bd=1, relief=SUNKEN)
separador.pack(fill=X, padx=5, pady=5)
Label(text="dos").pack()
mainloop()
```

Su representación gráfica queda:



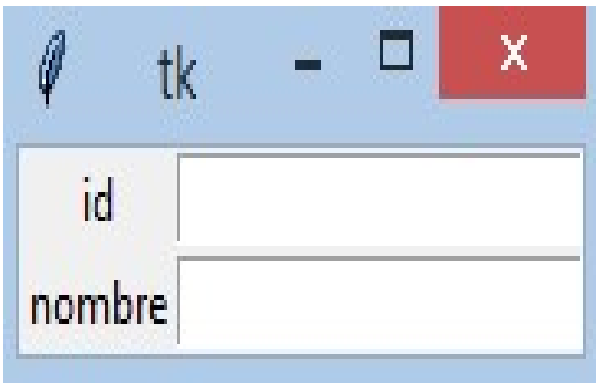
En el transcurso de las unidades y con la introducción de los bucles, estructuras de control y funciones iremos viendo cómo desarrollar aplicaciones más interesantes.

Grillas

El generador de grillas “Grid”, permite crear aplicaciones que se ubiquen en una tabla de dos dimensiones. El elemento padre es particionado en un determinado número de filas y columnas, y cada "celda" de la tabla resultante puede contener una nueva aplicación.

Es muy útil en la generación de formularios, por ejemplo para lograr crear un formulario con dos campos que posean dos entradas, lo podemos hacer de la siguiente forma:

```
from tkinter import *
root = Tk()
Label(root, text="id").grid(row=0, column=0)
Label(root, text="nombre").grid(row=1, column=0)
e1 = Entry(root)
e2 = Entry(root)
e1.grid(row=0, column=1)
e2.grid(row=1, column=1)
mainloop()
```



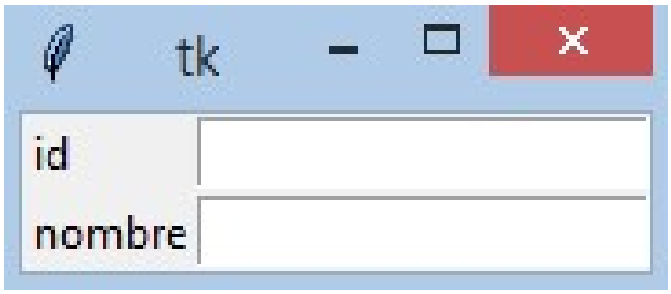
Uso de sticky

Dentro de una grilla, las aplicaciones se ubican centradas (esto se nota en el ejemplo anterior en el caso del Label que contiene el id), para modificar este comportamiento podemos utilizar el atributo "**sticky**", el cual toma como parámetros las opciones: N, S, W y E.

Así, si queremos que los nombres de los campos en el ejemplo anterior se posicionan a la izquierda, podemos modificar el código de la siguiente forma:

```
from tkinter import *
root = Tk()
Label(root, text="id").grid(row=0, column=0, sticky=W)
Label(root, text="nombre").grid(row=1, column=0, sticky=W)
e1 = Entry(root)
e2 = Entry(root)
e1.grid(row=0, column=1)
e2.grid(row=1, column=1)
mainloop()
```

Representación gráfica:



Botones y sus parámetros.

Cuando queremos agregar un botón, contamos con la estructura Button(), veamos un detalle de los parámetros que puede recibir.

Nomenclatura: Button(master=None, **options)

Función callback

command=nombreFunción

En este caso creamos un botón asociado al widget master con texto "text" y que ejecuta la función "callback", una función como veremos en la siguiente unidad es una rutina definida con un nombre al cual se le antepone la palabra reservada “def” y finaliza con dos puntos (:), y lo que hace la rutina se pone debajo pero agregando una tabulación o serie de espacios para indicar que pertenece a la función.

Nota: No nos preocupemos a esta altura de su estructura, simplemente veamos una aplicación para irnos acostumbrando a su existencia.

```
from tkinter import *

master = Tk()
def callback():
    print("click!")
b = Button(master, text="OK", command=callback)
b.pack()
mainloop()
```

Deshabilitarlo

Para en una construcción de prueba, poder deshabilitar el botón, le agrego state=DISABLED

```
b = Button(master, text="Help", state=DISABLED)
```

Padding

El padding es un espaciado interno al botón, que permite delimitar el área de escritura dentro del mismo.

```
from tkinter import *
master = Tk()
def callback():
    print("click!")
b = Button(master, text="OK", command=callback, padx=132, pady=132)
b.pack()
mainloop()
```

Alto y ancho

El alto y ancho de un botón de texto está dado en unidades de texto, mientras que si es una imagen, se trabaja en px.

```
from tkinter import *

master = Tk()

b = Button(master, text="Sure!", anchor=W, justify=LEFT, padx=22, height=3, width=12)
b.pack(fill=BOTH, expand=1)
mainloop()
```

Imagen

La imagen si no implementamos otra librería adicional debe estar en formato .gif

```
from tkinter import *
master = Tk()

photo=PhotoImage(file="download.gif")
c = Button(master, text="Sure!", anchor=W, justify=LEFT, image=photo )
c.pack(fill=BOTH, expand=1)
mainloop()
```

Color.

Fondo: background="black" ó bg="black"

Letra: foreground="red" ó fg="red"

Fondo activo: activebackground="green"

Letra activa: activeforeground="yellow",

Letra deshabilitado: disabledforeground="blue"

```
from tkinter import *

master = Tk()
def callback():
    print("click!")
b = Button(master, text="OK", command=callback, padx=132, pady=132,
activebackground="green", activeforeground="yellow",
background="black", foreground="red"
)
b.pack()
a = Button(master, text="OK", command=callback, padx=132, pady=132,
state=DISABLED, background="black", disabledforeground="blue"
)
a.pack()
mainloop()
```

Posición de texto/imagen y botón

Posición de texto e imagen

anchor=SW

Opciones: N, NE, E, SE, S, SW, W, NW, o CENTER. Por default es CENTER. (anchor/Anchor)

Posición de botón

Esta opción va dentro de pack()

side=LEFT

Opciones: LEFT, TOP, RIGHT, BOTTOM

```
from tkinter import *

master = Tk()
master.geometry("300x300")
def callback():
    print("click!")
b = Button(master, text="OK", command=callback,
    activebackground="green", activeforeground="yellow",
    background="black", foreground="red", height=7, width=12, anchor=SW)
b.pack(side=LEFT)
mainloop()
```

Tipo de texto

font=('tipo', tamaño, 'peso')

```
from tkinter import *

master = Tk()

b = Button(master, text="Ok!", anchor=W, justify=LEFT, padx=22,
height=3, width=12, font=('courier', 22, 'bold'))
b.pack(fill=BOTH, expand=1)
mainloop()
```

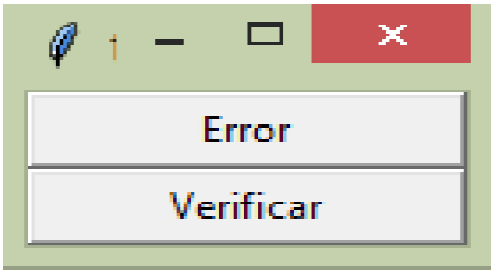
GUI – Funciones y funcionalidades.

Cajas de diálogo

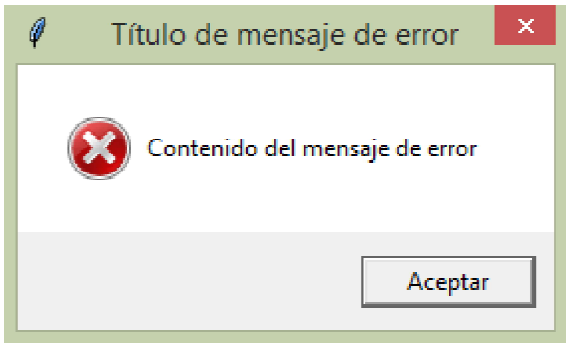
El módulo tkMessageBox de tkinter nos provee de una interface para representar cuadros de diálogo, mediante el uso de las funciones: showinfo, showwarning, showerror, askquestion, askokcancel, askyesno, o askretrycancel. El nombre de cada una de estas funciones ya nos dice para qué se utilizan, veamos un ejemplo de su implementación.

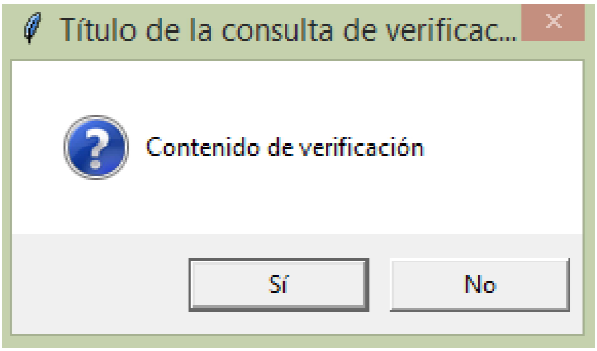
```
from tkinter import *
from tkinter.messagebox import *

def mensaje_error():
    showerror("Título de mensaje de error",
        "Contenido del mensaje de error")

def verificar():
    if askyesno('Título de la consulta de verificación',
        'Contenido de verificación'):
        showinfo('Si', 'Mensaje de información')
    else:
        showinfo('No', 'Esta a punto de salir')

Button(text='Error', command=mensaje_error).pack(fill=X)
Button(text='Verificar', command=verificar).pack(fill=X)

mainloop()
```

Representación gráfica pantalla principal:



Opciones de mensaje de Error y “Verificar”:





Podríamos modificar el script anterior con el uso de una función lambda para ejecutar la función error, según se muestra en el siguiente ejemplo:

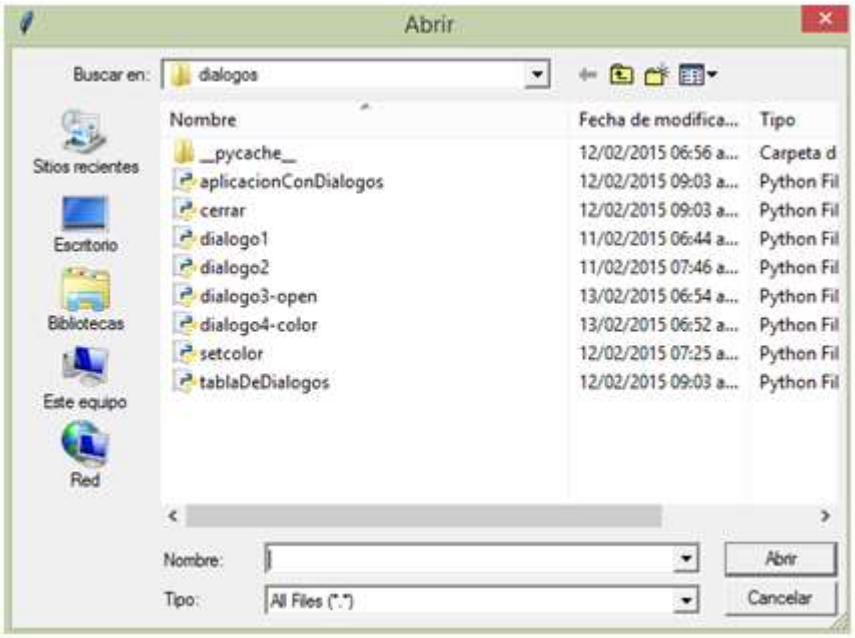
```
from tkinter import *
from tkinter.messagebox import *

def verificar():
    if askyesno('Título de la consulta de verificación',
        'Contenido de verificación'):
        showinfo('Si', 'Mensaje de información')
    else:
        showinfo('No', 'Esta a punto de salir')
Button(text='Error', command=(lambda: showerror('Título de mensaje de
error',"Contenido del mensaje de error"))).pack(fill=X)
Button(text='Verificar', command=verificar).pack(fill=X)

mainloop()
```

Cajas de diálogo – Seleccionar Archivo

En determinada ocasiones necesitamos poder seleccionar archivos y abrirlos para leerlos o editarlos, para estos casos tkinter nos proporcionar el método askopenfilename () con el cual podemos seleccionar un archivo y obtener su path. El siguiente ejemplo presenta un botón que al presionarlo nos permite seleccionar un archivo e imprimir su path en pantalla.

```
from tkinter import *
from tkinter.filedialog import askopenfilename


def callback():
    ruta = askopenfilename()
    print(ruta)

Button(text='Abrir archivo', command=callback).pack(fill=X)
mainloop()
```

Visualización:

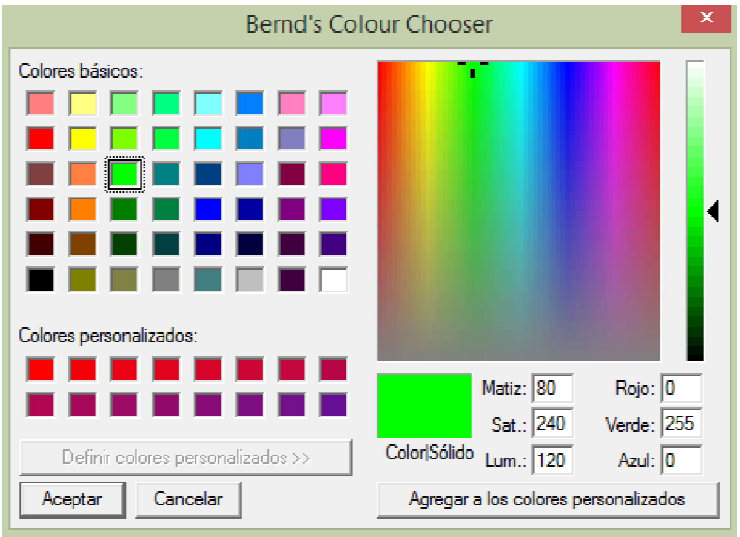


Cajas de diálogo – Selección de color

Otro método muy útil es askcolor (), con el cual podemos seleccionar desde una paleta de colores un determinado color, veamos cómo utilizarlo con el siguiente ejemplo:

```
from tkinter import *
from tkinter.colorchooser import askcolor

def callback():
    result = askcolor(color="#00ff00",title = "Bernd's Colour
          Chooser")
    print(result)
    print(result[1])

root = Tk()
Button(root, text='Seleccionar color', fg="green",
    command=callback).pack(side=LEFT, padx=10)
Button(text='Cerrar', command=root.quit,
    fg="red").pack(side=LEFT, padx=10)
mainloop()
```

Visualización:



Imágenes

En tkinter, las imágenes pueden ser presentadas dentro de botones, cajas, canvas, etc, al asociarles una imagen o un bitmap mediante el uso de un atributo. En el siguiente ejemplo vemos como dada la ruta a un directorio donde se encuentra una imagen, podemos mediante el método PhotoImage() asociarle la imagen a un botón con el uso del atributo image. Dado que no hemos especificados dimensiones para el botón, este adquiere la dimensión de la imagen.

```
from tkinter import *
ruta = "img/"
win = Tk()
imagen = PhotoImage(file=ruta + "download.gif")
Button(win, image=imagen).pack()
win.mainloop()
```

Visualización:

Como podemos ver a continuación, es posible utilizar otro elemento de tkinter, en este caso un canvas.

```
from tkinter import *
ruta = "img/"
win = Tk()
imagen = PhotoImage(file=ruta + "download.gif")
can = Canvas(win)
can.pack(fill=BOTH)
```

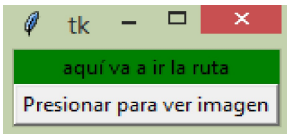
```
can.create_image(2, 2, image=imagen, anchor=NW)
win.mainloop()
```

Seleccionar imagen

Vamos ahora a introducir el módulo ***glob***, el cual nos permite seleccionar una serie de archivos que tengan una determinada extensión, y lo vamos a utilizar para presentar una imagen de forma aleatoria seleccionada de un determinado directorio (mediante el uso del módulo random) al presionar en un botón.

```
from tkinter import *
from glob import glob
import random
ruta = 'img/'

def seleccion():
    nombre, foto = random.choice(imagen)
    dialogo.config(text=nombre)
    boton.config(image=foto)

root=Tk()
dialogo = Label(root, text="aquí va a ir la ruta", bg='OrangeRed')
boton = Button(root, text="Presionar para ver imagen", command=seleccion)
dialogo.pack(fill=BOTH)
boton.pack()

archivo = glob(ruta + "*.gif")
imagen = [(x, PhotoImage(file=x)) for x in archivo]
print(archivo)
root.mainloop()
```

Visualización:



Librería PILLOW

Para poder trabajar con una mayor cantidad de formatos de imágenes e incluso editarlas, podemos utilizar la librería de python PILLOW (que es una actualización de la librería PIL) la cual nos permite trabajar con más de 30 formatos, entre los cuales se encuentran por ejemplo: png, jpeg, jpg.

La librería no tiene porqué utilizarse dentro de tkinter, pero si la integramos a tkinter, ya viene con métodos que poseen el mismo nombre que los que tkinter utiliza de forma que su integración requiera únicamente realizar la importación del método correspondiente, en este caso el método PhotoImage de PILLOW el cual pisa al método PhotoImage de tkinter, ya que este únicamente nos permite trabajar con formatos .gif.

```
from tkinter import *
from PIL.ImageTk import PhotoImage
from glob import glob
import random
ruta = 'images/'

def seleccion():
    nombre, foto = random.choice(imagen)
    dialogo.config(text=nombre)
    boton.config(image=foto)

root=Tk()
dialogo = Label(root, text="aquí va a ir la ruta", bg='green')
boton = Button(root, text="Presionar para ver imagen", command=seleccion)
dialogo.pack(fill=BOTH)
boton.pack()
```

```
archivo = glob(ruta + "*.jpg")
imagen = [(x, PhotoImage(file=x)) for x in archivo]
print(archivo)
root.mainloop()
```

Menú

Una herramienta fundamental para lograr realizar una aplicación ordenada, es el widget Menu, el cual nos permite crear menús desplegables y popups.

La forma de utilizarlo, es creando un elemento menú, el cual está asociado a una caja

```
menubar = Menu(root)
```

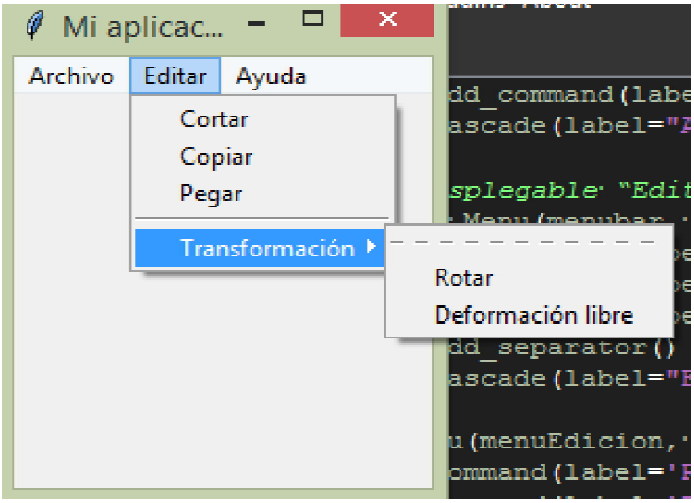
y luego crear un elemento del menú y agregarle los elementos que vamos a utilizar (mediante **.add_command()**), cada elemento puede ser asociado a una función (en este caso todos están asociados a la función "**hola**" que lo único que hace es imprimir un "Hola!" en pantalla. El elemento es asociado al menú mediante "**.add_cascade()**", e incluso podemos agregar un separador entre elementos mediante "**add_separator()**"

```
menuArchivo = Menu(menubar, tearoff=0)
menuArchivo.add_command(label="Abrir", command=hola)
menuArchivo.add_command(label="Guardar", command=hola)
menuArchivo.add_separator()
menuArchivo.add_command(label="Salir", command=root.quit)
menubar.add_cascade(label="Archivo", menu=menuArchivo)
```

En caso de querer agregar un submenú, podemos realizarlo, asociando el submenú al elemento correspondiente:

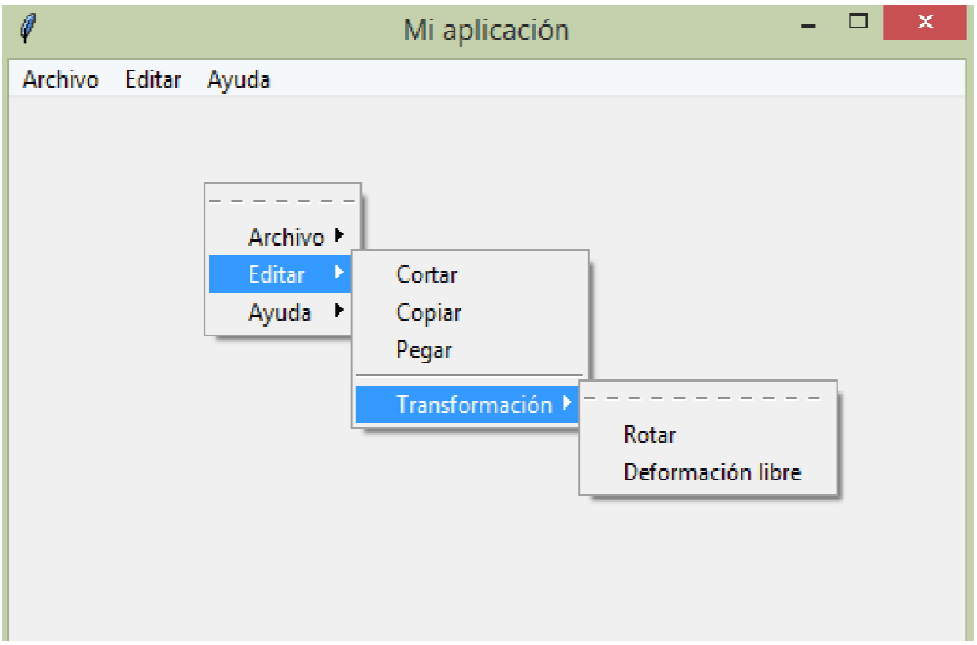
```
submenu = Menu(menuEdicion, tearoff=True)
submenu.add_command(label='Rotar', command=hola)
submenu.add_command(label='Deformación libre', command=hola)
menuEdicion.add_cascade(label='Transformación', menu=submenu)
```

Representación gráfica:



Menú - popups

Algo muy útil y a lo cual estamos acostumbrados en muchas aplicaciones, es al uso de popups para mostrar los menús, como puede verse en la siguiente representación:



En la mayoría de las aplicaciones, esto se realiza mediante el uso del botón derecho del mouse sobre el área de trabajo (en tkinter el botón derecho viene representado por: **<Button-3>**). El método bind() es en este caso el que utilizamos para crear el popup, asociando en este caso la función "popup()" y ejecutando dentro de la misma, una línea de código en donde indicamos mediante "post()" que un determinado menú sea

representado a partir de las coordenadas x e y en donde presionamos con el botón derecho del mouse.

```
#####################################################################
# crear una caja
#####################################################################
frame = Frame(root, width=512, height=512)
frame.pack()

def popup(event):
    menubar.post(event.x_root, event.y_root)

# asociar el popup a la caja
frame.bind("<Button-3>", popup)
```

Nota: Ver código completo en menu2-popup.py

Menú - Reconfiguración

Algo que es muy útil, es poder llevar un registro de eventos pasados, como puede verse en la herramienta historial de Photoshop, para esto le pasamos al parámetro "**postcommand**" una determinada función que hace algo, en el ejemplo siguiente es asociada a la función "update()", la cual cada vez que es invocada incrementa el valor de la variable global "**contar**" en una unidad.

```
contar = 0

def update():
    global contar
    contar = contar + 1
    menuPrueba.entryconfig(0, label=str(contar))

menuPrueba = Menu(menubar, tearoff=0, postcommand=update)
menuPrueba.add_command(label=str(contar))
menuPrueba.add_command(label="Salir", command=root.quit)
menubar.add_cascade(label="Prueba", menu=menuPrueba)
```

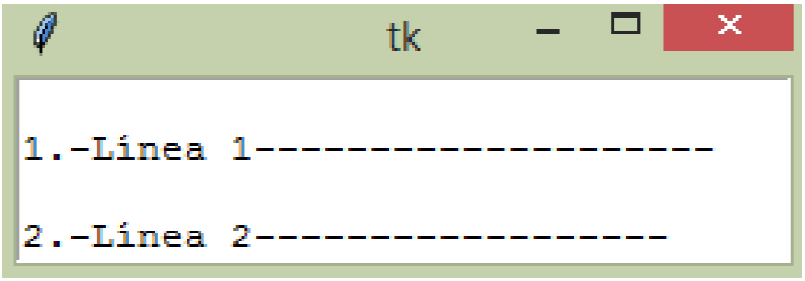
Nota: Ver código completo en menu3-reconfigurar.py

Texto

Para ingresar por pantalla grandes cantidades de texto, de forma que posteriormente se pueda analizar, utilizamos el widget Text(), su utilización es similar al resto de los widgets, en este caso para agregar texto lo logramos mediante el método **"insert()"**. La cantidad de líneas de texto a visualizar está representado por el atributo **"height"**, y la cantidad de caracteres por **"width"**

```
from tkinter import *
root = Tk()

T = Text(root, height=4, width=33)
T.pack()
texto1 = """
1.-Línea 1----------
\n2.-Línea 2----------
\n3.-Línea 3----------
\n4.-Línea 4----------
\n5.-Línea 5----------
\n6.-Línea 6----------
"""
T.insert(END, texto1)
mainloop()
```

Representación gráfica:

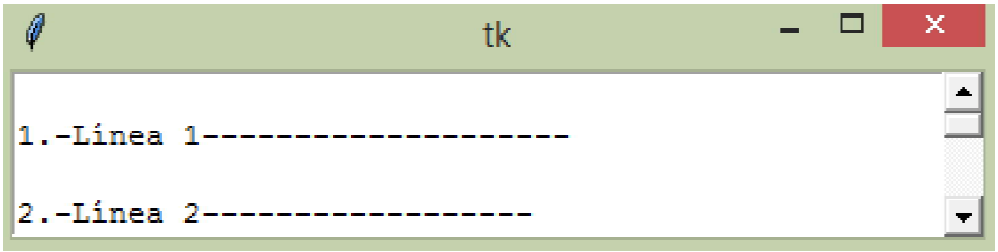


Menú - Scrollbar

En caso de que queramos que el área de trabajo permita visualizar más de las líneas preestablecidas, podemos utilizar el widget "**Scrollbar()**" como se muestra a continuación

```
from tkinter import *
root = Tk()
S = Scrollbar(root)
T = Text(root, height=4, width=50)
S.pack(side=RIGHT, fill=Y)
T.pack(side=LEFT, fill=Y)
S.config(command=T.yview)
T.config(yscrollcommand=S.set)
texto1 = """
1.-Línea 1----------
\n2.-Línea 2----------
\n3.-Línea 3----------
\n4.-Línea 4----------
\n5.-Línea 5----------
\n6.-Línea 6----------
"""
T.insert(END, texto1)
mainloop( )
```

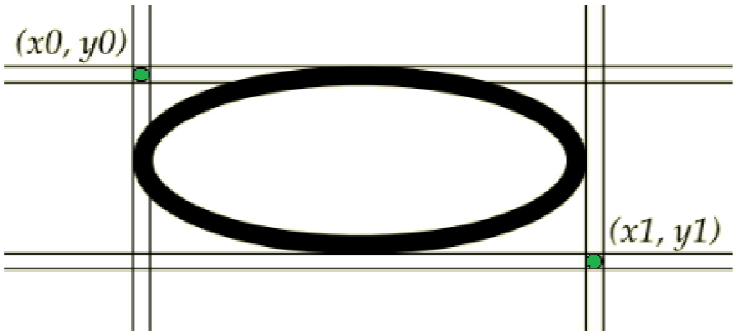
Representación gráfica:



Canvas

En caso de que queramos implementar una aplicación tipo “Paint” necesitamos trabajar con el widget “**Canvas()**”, en el cual podemos agregar imágenes y figuras como pueden ser arcos, polígonos, etc.

Para agregar un óvalo, debemos tener presente cómo se establecen sus parámetros, los dos primeros parámetros de “**create_oval**” son (x0, y0), los siguiente dos son (x1, y1) y luego vienen los parámetros para configurar espesor de línea, color, etc



Las siguientes líneas muestran una representación de tres óvalos.

```
canvas.create_oval(70, 100, 140, 200, width=2, fill='blue')
canvas.create_oval(80, 140, 120, 200, width=2, fill='white')
canvas.create_oval(90, 160, 110, 180, width=2, fill='black')
```

Para agregar una línea, le pasamos como parámetros a “create_line”:

canvas.create_line(x0, y0, x1, y1, otros parámetros)

```
canvas.create_line(200, 100, 300, 200, width=3, fill='OrangeRed')
```

Para agregar un polígono, introducimos sus puntos de a pares

```
canvas.create_polygon(165, 165, 155, 220,180, 220, width=2, fill='green')
```

Quizás el elemento más complejo de representar es el arco, en donde los cuatro primeros parámetros se utilizan análogamente que en el óvalo, luego determinamos el ángulo inicial del arco con start (en grados) y el final con “extent”

```
canvas.create_arc(60, 60, 140, 100, start=0,
extent=210, outline="#f11", fill="#1f1", width=2)
canvas.create_arc(200, 60, 280, 100, start=-20,
extent=200, outline="#f11", fill="#1f1", width=2)
```

La siguiente es la representación del ejemplo “canvas1.py”, en donde el contorno es de una imagen en formato. gif: