RESUMEN INTERFAZ GRÁFICA

PRIMERA PARTE

3 Noviembre del 2010

Autor: Grupo 50, Segundo Semestre 2010 (Sede San Carlos)

1. Introducción:

Este tutorial no está hecho para seguirlo al pie de la letra solo son algunas cosas básicas que debemos entender para poder trabajar con ventanas y para dar un pequeño salto desde la cónsola a la interfaz. Recordemos que somos la primera generación en el TEC que recibe Python, por lo que el tutorial es meramente experimental, y puede contener cosas que no son del todo correctas. Se tratará de mostrar cómo se trabaja desde ciertos puntos de vista, y el usuario se encargará de desarrollar su propia forma de construir la interfaz.

2. El Módulo que usaremos

Un módulo se puede importar en Python. Existen diferentes formas de crear módulos, y de importarlos. Un módulo puede ser un archivo .py, o todo un conjunto de archivos que interactúan con el sistema operativo y la máquina en general.

Por ejemplo el módulo time obtiene datos sobre la fecha, el módulo os obtiene datos sobre el sistema operativo.

Python ya viene con varios módulos integrados, sin embargo nosotros podemos buscar en internet y descargar otros más.

Para construir interfaz gráfica en Python se pueden utilizar varios módulos. Se pueden descargar módulos para crear la interfaz de la forma que más nos guste.

Para este manual, usaremos el módulo Tkinter, porque ya viene instalado en Python y es con el que hemos trabajado, y
bastará para darse una idea de cómo se trabaja en Python con interfaz gráfica.

En algunas versiones de Linux no viene instalado el módulo Tkinter, entonces debemos escribir en la cónsola:
sudo apt-get install python-tk

Mientras que la versión que uno se descarga para Windows ya lo trae.

3. El IDE (El software que usaremos para programar)

Bien, ahora necesitamos un IDE, que es donde programaremos. Yo personalmente he utilizado Eclipse, Netbeans, el que trae
Python por defecto, uno que se llama Pycharm que es de pago (Lo cual en cierto modo representa un problema) y al final he
dado con uno que en lo personal me agrada y se llama Pyscripter. La verdad me parece la alternativa más indicada para
programar en Python (Tomando en cuenta que no realizaremos un super programa empresarial, si no simples prácticas)

En google pueden buscar y encontrarán versiones para varios sistemas operativos.

Si no, pueden utilizar cualquiera que gusten.

4. Importando el módulo

Para importar un módulo en Python escribimos

```
import nombredelmodulo
```

Otra forma de hacerlo es

```
from nombredelmodulo import *
```

Esto no les va a devolver nada porque el modulo nombredelmodulo no existe. Pero básicamente estas son las formas que se usan más.

5. Ventanas

Las ventanas se pueden configurar de diferentes formas. Veamos un ejemplo de una ventana.

Ejemplo 1:

```
from Tkinter import * # Importa el módulo

v0 = Tk() # Tk() Es la ventana principal

v0.mainloop() # Es el evento que llama al inicio de nuestro programa. Siempre lo lleva la ventana principal.
```

Esto nos lanza una ventana normal.

Ejemplo 2:

Ahora para colocarle fondo a la ventana sería:

```
from Tkinter import * # Importa el módulo

v0 = Tk() # Tk() Es la ventana principal
v0.config(bg="black") # Le da color al fondo

v0.mainloop() # Es el evento que llama al inicio de nuestro programa.
Siempre lo lleva la ventana principal.
```

Ejemplo 3:

Una forma de controlar el tamaño de la ventana sería el siguiente:

```
from Tkinter import * # Importa el módulo

v0 = Tk() # Tk() Es la ventana principal
v0.config(bg="black") # Le da color al fondo
v0.geometry("500x500") # Cambia el tamaño de la ventana

v0.mainloop() # Es el evento que llama al inicio de nuestro programa.
Siempre lo lleva la ventana principal.
```

Ejemplo 4:

Bueno como habrán visto, declaramos una variable llamada v0, del tipo Tk().

Además de Tk(), podemos declarar más variables

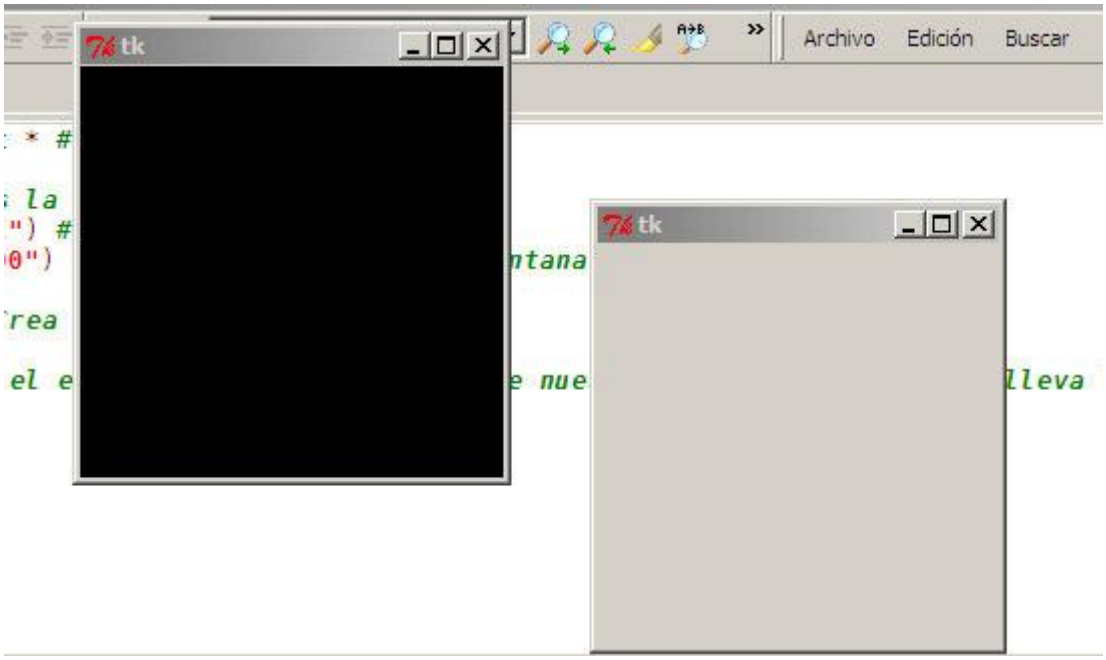
Toplevel: Crea una nueva ventana
Frame: Coloca los páneles para ordenar los elementos
Canvas: Para dibujar y graficar funciones etc..
Button: Para colocar un botón
Label: Coloca un texto
Message: Coloca un texto
Entry: Coloca una entrada de texto de una linea
Text: Coloca una entrada de texto de varias lineas
Listbox: Coloca una lista con elementos clickeables
Menú: Coloca un menú que puede contener cascadas y elementos clickeables

Existe toda una mecánica para agregar elementos a una ventana. Además de todo esto se pueden colocar imágenes y otras
cosas. Por el momento estos controles básicos son los que aprenderemos a utilizar:

Ejemplo 5:

Vamos a crear una ventana de tipo Toplevel. En Python ocurre algo curioso, todas las ventanas creadas se abren al mismo
tiempo. Por eso cuando creamos una ventana del tipo Toplevel, debemos ocultarla. Veamos un ejemplo:

```
from Tkinter import * # Importa el módulo

v0 = Tk() # Tk() Es la ventana principal
v0.config(bg="black") # Le da color al fondo
v0.geometry("500x500") # Cambia el tamaño de la ventana

v1=Toplevel(v0) # Crea una ventana hija

v0.mainloop() # Es el evento que llama al inicio de nuestro programa.
Siempre lo lleva la ventana principal.
```



Se abre al mismo tiempo las dos ventanas. Esto se evita de la siguiente forma:

```
from Tkinter import * # Importa el módulo

v0 = Tk() # Tk() Es la ventana principal
v0.config(bg="black") # Le da color al fondo
v0.geometry("500x500") # Cambia el tamaño de la ventana

v1=Toplevel(v0) # Crea una ventana hija

v1.withdraw()  # oculta v1
v0.mainloop() # Es el evento que llama al inicio de nuestro programa. Siempre lo lleva la ventana principal.
```

Ahí ya ocultó la ventana. Pero cómo mostrarla otra vez?

Ejemplo 6:

Para importar un botón se hace de la siguiente forma:

```
from Tkinter import * # Importa el módulo

v0 = Tk() # Tk() Es la ventana principal
v0.config(bg="black") # Le da color al fondo
v0.geometry("500x500") # Cambia el tamaño de la ventana
b1=Button(v0,text="ABRIR VENTANA V1") # Primer botón
b1.pack() # El botón es cargado

v1=Toplevel(v0) # Crea una ventana hija

v1.withdraw() # Oculta la ventana v1
v0.mainloop() # Es el evento que llama al inicio de nuestro programa.
Siempre lo lleva la ventana principal.
```

Ese botón no tiene ningún evento, además fue cargado con el método .pack. Lo podemos cargar con el método .grid

Son dos formas distintas de cargar un botón. En vez de la primera forma:

```
b1=Button(v0,text="ABRIR VENTANA V1") # Primer botón
b1.pack() # El botón es cargado
```

Pudimos haberlo escrito de esta otra forma:

```
b1=Button(v0,text="ABRIR VENTANA V1").pack()
```

En una sola línea. O de esta otra forma:

```
b1=Button(v0,text="ABRIR VENTANA V1") # Primer botón
b1.grid(row=1,column=1) # El botón es cargado
```

Y esta otra forma:

```
b1=Button(v0,text="ABRIR VENTANA V1").grid(row=1,column=1) # El botón es
cargado
```

Hay diversas formas de cargar los botones y que se ajustan a nuestro gusto y forma de hacer las cosas.

Ahora vamos a mostrar la otra ventana.

Primero creamos una función llamada mostrar

```
def mostrar(ventana):
        ventana.deiconify()
```

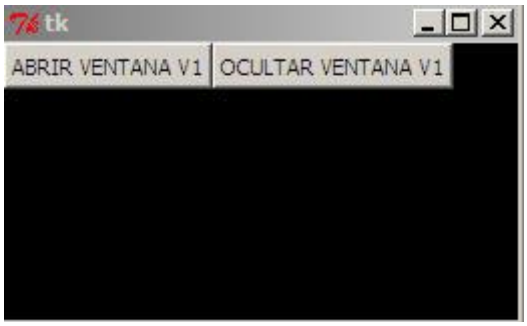
El método deiconify muestra la ventana que había sido oculta por withdraw(). Además una función para ocultar podría ser

```
def ocultar(ventana):
        ventana.withdraw()
```

Con solo estas dos funciones ya podemos mostrar y ocultar la ventana 1. vamos a agregar otro botón más y todo junto queda
de la siguiente forma:

```
from Tkinter import * # Importa el módulo

v0 = Tk() # Tk() Es la ventana principal
v1=Toplevel(v0) # Crea una ventana hija

def mostrar(ventana): ventana.deiconify()
def ocultar(ventana):ventana.withdraw()
def ejecutar(f): v0.after(200,f)

v0.config(bg="black") # Le da color al fondo
v0.geometry("500x500") # Cambia el tamaño de la ventana

b1=Button(v0,text="ABRIR VENTANA V1",command=lambda:
ejecutar(mostrar(v1))) # Primer botón
b1.grid(row=1,column=1) # El botón es cargado
b2=Button(v0,text="OCULTAR VENTANA V1",command=lambda:
ejecutar(ocultar(v1))) # Segundo botón
b2.grid(row=1,column=2) # El botón es cargado
```

```
v1.withdraw() # Oculta la ventana v1
v0.mainloop() # Es el evento que llama al inicio de nuestro programa.
Siempre lo lleva la ventana principal.
```



El próximo archivo comentaremos algunos aspectos a tomar en cuenta a la hora de mostrar y ocultar ventanas

RESUMEN INTERFAZ GRÁFICA

SEGUNDA PARTE

3 Noviembre del 2010

Autor: Grupo 50, Segundo Semestre 2010 (Sede San Carlos)

6. Irregularidades

Una de las cosas que dificulta a una persona nueva que quiere aprender interfaz en Python, es el conjunto de dificultades
que aparecen conforme vas aprendiendo, sobre todo si no dominas bien más de un lenguaje o solo haz programado en Scheme
una interfaz gráfica y piensas que todas se programan igual. Pues presenta una dificultad.
En el último código que vimos:

```
from Tkinter import * # Importa el módulo

v0 = Tk() # Tk() Es la ventana principal
v1=Toplevel(v0) # Crea una ventana hija
```

```
def mostrar(ventana): ventana.deiconify() # Muestra una ventana
def ocultar(ventana):ventana.withdraw() # Oculta una ventana
def ejecutar(f): v0.after(200,f) # Una forma alternativa de ejecutar una
función

v0.config(bg="black") # Le da color al fondo
v0.geometry("500x500") # Cambia el tamaño de la ventana

b1=Button(v0,text="ABRIR VENTANA V1",command=lambda:
ejecutar(mostrar(v1))) # Primer botón
b1.grid(row=1,column=1) # El botón es cargado
b2=Button(v0,text="OCULTAR VENTANA V1",command=lambda:
ejecutar(ocultar(v1))) # Segundo botón
b2.grid(row=1,column=2) # El botón es cargado

v1.withdraw() # Oculta la ventana v1
v0.mainloop() # Es el evento que llama al inicio de nuestro programa.
Siempre lo lleva la ventana principal.
```

Vemos que apareció:
Una nueva función llamada ejecutar
¿v0.after? ¡¿Qué es eso?!
¿Porqué el command de los botones tiene lambda dos puntos ejecutar y la función dentro de mostrar?

Esto fue así, porque hice varias pruebas y esta fué la mejor forma que encontré para que todo funcionara bien (Se testeó la
interfaz en Windows XP).

Otra cosa que noté es que la interfaz de vez en cuando no actúa de la misma forma en Windows que en las distros de Linux
(Por ejemplo Ubuntu o Kubuntu).

Ya por ejemplo si yo quiero centrar una ventana de Python en windows, utilizo esta función:

```
def centrar(ventana):
    ventana.update_idletasks()
    w=ventana.winfo_width()
    h=ventana.winfo_height()
    extraW=ventana.winfo_screenwidth()-w
    extraH=ventana.winfo_screenheight()-h
    ventana.geometry("%dx%d%+d%+d" % (w,h,extraW/2,extraH/2))
```

Si escribimos el nombre de la ventana, entonces la centrará, pero en Windows, porque en Ubuntu desconfigura todo el tamaño de la ventana.

En cuánto a las preguntas de más arriba. La función llamada ejecutar, y lo de v0.after, es acerca de lo mismo. ¿Qué es lo que hace la ventana ejecutar?

Lo que hace es que pasados dos milisegundos en un momento dado del tiempo, ejecuta f. f es el nombre de una función que queremos ejecutar.

Por ejemplo abre una ventana, como se vió anteriormente.

Y para qué es el lambda? No estoy muy seguro pero es absolutamente necesario usarlo, además lambda permite que un botón ejecute dos eventos al mismo tiempo.

Ejemplo 7:

Por ejemplo si queremos mostrar una ventana y también imprimir hola, sería de la siguiente forma:

```
from Tkinter import * # Importa el módulo

v0 = Tk() # Tk() Es la ventana principal
v1=Toplevel(v0) # Crea una ventana hija
```

```
def mostrar(ventana): ventana.deiconify() # Muestra una ventana
def ocultar(ventana):ventana.withdraw() # Oculta una ventana
def ejecutar(f): v0.after(200,f) # Una forma de ejecutar las funciones
def imprimir(texto): print texto # Imprime un texto

v0.config(bg="black") # Le da color al fondo
v0.geometry("500x500") # Cambia el tamaño de la ventana

b1=Button(v0,text="ABRIR VENTANA V1",command=lambda:
ejecutar(mostrar(v1)) or imprimir("hola")) # Primer botón
b1.grid(row=1,column=1) # El botón es cargado
b2=Button(v0,text="OCULTAR VENTANA V1",command=lambda:
ejecutar(ocultar(v1))) # Segundo botón
b2.grid(row=1,column=2) # El botón es cargado

v1.withdraw() # Oculta la ventana v1
v0.mainloop() # Es el evento que llama al inicio de nuestro programa.
Siempre lo lleva la ventana principal.
```

Ejemplo 8:

Si dan click al botón ABRIR VENTANA V1, notarán que además de mostrar la ventana, imprime el texto. Si quisieramos que el
botón hiciera tres cosas, sería de la siguiente forma:

```
from Tkinter import * # Importa el módulo

v0 = Tk() # Tk() Es la ventana principal
v1=Toplevel(v0) # Crea una ventana hija

def mostrar(ventana): ventana.deiconify() # Muestra una ventana
def ocultar(ventana):ventana.withdraw() # Oculta una ventana
def ejecutar(f): v0.after(200,f) # Una forma de ejecutar las funciones
def imprimir(texto): print texto # Imprime un texto

v0.config(bg="black") # Le da color al fondo
```

```
v0.geometry("500x500") # Cambia el tamaño de la ventana

b1=Button(v0,text="ABRIR VENTANA V1",command=lambda:
ejecutar(mostrar(v1)) or imprimir("hola") or imprimir("tercera función"))
# Primer botón
b1.grid(row=1,column=1) # El botón es cargado
b2=Button(v0,text="OCULTAR VENTANA V1",command=lambda:
ejecutar(ocultar(v1))) # Segundo botón
b2.grid(row=1,column=2) # El botón es cargado

v1.withdraw() # Oculta la ventana v1
v0.mainloop() # Es el evento que llama al inicio de nuestro programa.
Siempre lo lleva la ventana principal.
```

Con el lambda y los dos puntos, mezclado con la función ejecutar, podemos ejecutar varios eventos al mismo tiempo separados por **or**.

Ejemplo 9:

Ahora, no sé si lo habrán notado, pero cuando cerramos la ventana, tira error, pero cuando la ocultamos no nos da error. Yo la solución que le encontré a esto, fue quitando la opción de cerrar ventana, y colocando un botón salir en cada ventana. Para realizar eso se hace lo siguiente:

```
from Tkinter import * # Importa el módulo

v0 = Tk() # Tk() Es la ventana principal
v1=Toplevel(v0) # Crea una ventana hija
v1.protocol("WM_DELETE_WINDOW", "onexit") # Elimina la opción de salir
para evitar el error

def mostrar(ventana): ventana.deiconify() # Muestra una ventana
def ocultar(ventana):ventana.withdraw() # Oculta una ventana
def ejecutar(f): v0.after(200,f) # Una forma de ejecutar las funciones
```

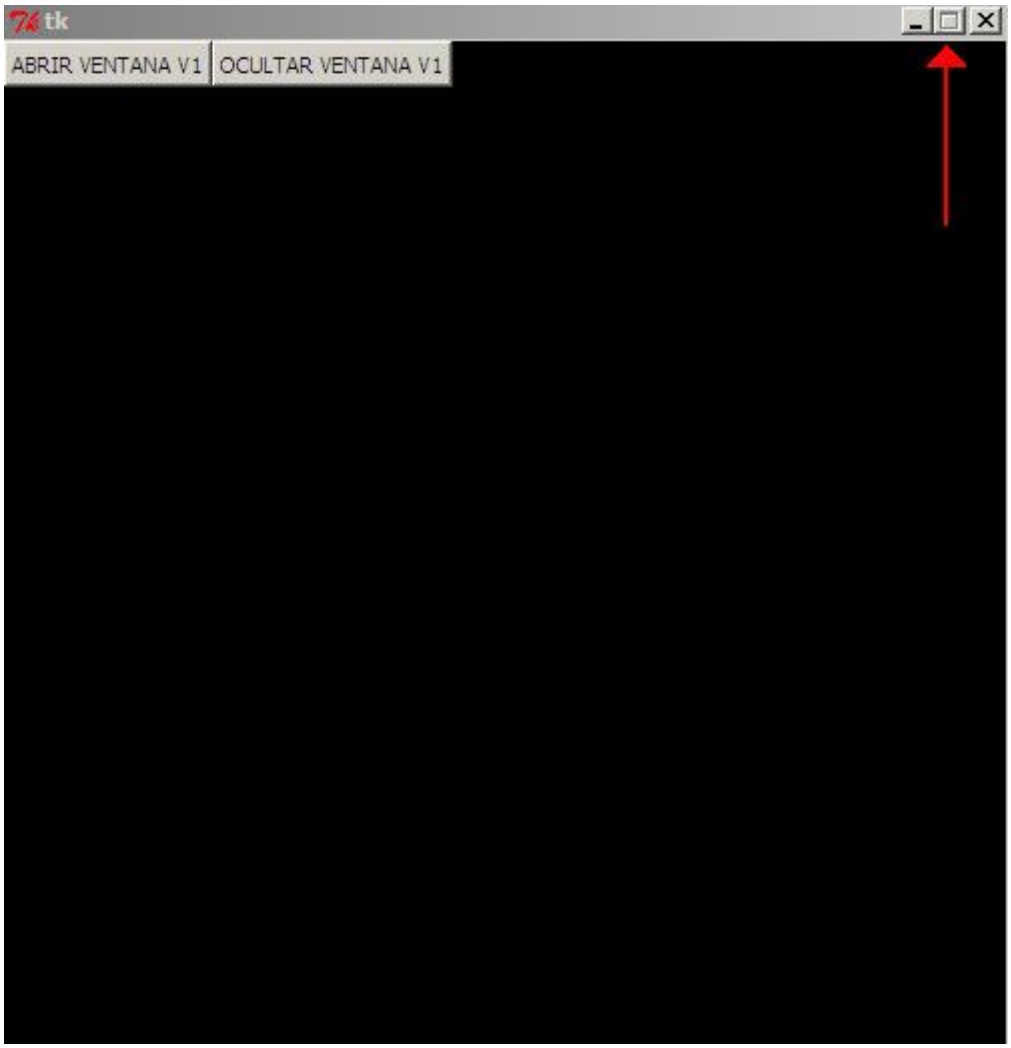
```
def imprimir(texto): print texto # Imprime un texto

v0.config(bg="black") # Le da color al fondo
v0.geometry("500x500") # Cambia el tamaño de la ventana

b1=Button(v0,text="ABRIR VENTANA V1",command=lambda:
ejecutar(mostrar(v1)) or imprimir("hola") or imprimir("tercera función"))
# Primer botón
b1.grid(row=1,column=1) # El botón es cargado
b2=Button(v1,text="OCULTAR VENTANA V1",command=lambda:
ejecutar(ocultar(v1))) # Segundo botón
b2.grid(row=1,column=2) # El botón es cargado

b3=Button(v0,text="SALIR",command=lambda: ejecutar(ocultar(v1)))
b3.grid(row=1,column=2) # El botón es cargado

v1.withdraw() # Oculta la ventana v1
v0.mainloop() # Es el evento que llama al inicio de nuestro programa.
Siempre lo lleva la ventana principal.
```

Ejemplo 10:

Otro truco interesante es evitar que una ventana pueda redimencionarse (Cambiar el tamaño de una ventana). Con lo siguiente se puede realizar:

```
from Tkinter import * # Importa el módulo

v0 = Tk() # Tk() Es la ventana principal
v1=Toplevel(v0) # Crea una ventana hija
v1.protocol("WM_DELETE_WINDOW", "onexit") # Elimina la opción de salir
para evitar el error
v0.resizable(0,0) # Evita que la ventana se pueda cambiar de tamaño
v1.resizable(0,0) # Evita que se le pueda cambiar de tamaño a la ventana

def mostrar(ventana): ventana.deiconify() # Muestra una ventana
```

```
def ocultar(ventana):ventana.withdraw() # Oculta una ventana
def ejecutar(f): v0.after(200,f) # Una forma de ejecutar las funciones
def imprimir(texto): print texto # Imprime un texto

v0.config(bg="black") # Le da color al fondo
v0.geometry("500x500") # Cambia el tamaño de la ventana

b1=Button(v0,text="ABRIR VENTANA V1",command=lambda:
ejecutar(mostrar(v1)) or imprimir("hola") or imprimir("tercera función"))
# Primer botón
b1.grid(row=1,column=1) # El botón es cargado
b2=Button(v1,text="OCULTARME",command=lambda: ejecutar(ocultar(v1))) #
Segundo botón
b2.grid(row=1,column=2) # El botón es cargado

b3=Button(v0,text="OCULTAR VENTANA V1",command=lambda:
ejecutar(ocultar(v1)))
b3.grid(row=1,column=2) # El botón es cargado

v1.withdraw() # Oculta la ventana v1
v0.mainloop() # Es el evento que llama al inicio de nuestro programa.
Siempre lo lleva la ventana principal.
```



Con una lectura de los manuales de Tkinter se deberían poder solucionar algunas de estas irregularidades, entre otras más
que se podrían comentar más adelante.

7. ELEMENTOS PADRES Y ELEMENTOS HIJOS

Tomando en cuenta todos los primeros 6 puntos, ahora vamos a analizar la interfaz por el lado de lo que son los elementos
hijos de otros, cuál es el padre de un elemento, etcétera. Generalmente cuando trabajamos con Tkinter, el elemento padre
es el primer argumento que recibe un objeto gráfico. El elemento Tk() no tiene ningún padre. Si declaramos una variable

v0 = Tk()

Entonces v0 es un Tk(), que es la ventana principal, y entre paréntesis no recibe ningún argumento. Pero si queremos
introducir un elemento dentro de v0, dicho elemento debe saber que su padre es v0.

En el siguiente ejemplo se crean dos ventanas, la ventana principal del tipo Tk() y una ventana hija del tipo Toplevel().

La ventana Toplevel() es hija de la ventana Tk()
Luego creamos un botón en cada ventana. Lo que querría decir que uno de los botones tiene como padre a la ventana Tk() y el

otro botón tiene como padre a la ventana del tipo Toplevel()

Ejemplo 11:

A continuación se presenta el ejemplo

```
from Tkinter import *

v0=Tk()
v1=Toplevel(v0)

def mostrar(ventana): ventana.deiconify()
def ocultar(ventana):ventana.withdraw()
def ejecutar(f): v0.after(200,f)

b1=Button(v0,text="TERMINAR APLICACIÓN",command=v0.destroy).pack()
b2=Button(v1,text="OCULTAR VENTANA",command=lambda:
ejecutar(ocultar(v1))).pack()

v0.mainloop()
```

En este ejemplo se puede ver que para salir de la aplicación basta con destruir el objeto de la ventana principal, llamado v0. Así, cuando escribimos **v0.destroy** se destruye la ventana padre, y entonces la aplicación termina.

Otras formas que podrían presentar conflictos, pero que también son usadas, son **v0.quit**, y por supuesto, **ocultar(v0)** que sería lo mismo que decir v0.withdraw(). Sin embargo con el **withdraw()** solo la oculta no cerraría la aplicación.

También se puede usar en el command del botón **lambda: exit()**

Ahora introduzcamos más controles, como por ejemplo los Labels

8. QUE EL PROGRAMA TAMBIÉN SEA UN OBJETO

Ejemplo 12:

Creamos un programa que funcione con el paradigma de objetos y se puede implementar de la siguiente forma

```
from Tkinter import *

class programa:
    def __init__(self):
        self.v0 = Tk()
    def run(self):
        self.v0.mainloop()
```

```
test1=programa()
test1.run()
```

Se implementa todo lo visto anteriormente pero a manera de objetos.

9. PROPIEDADES DE LOS CONTROLES (Parte 1)

Ya habíamos hablado de las ventanas Tk y Toplevel y de los controles Label, Button, Listbox, Scrollbar Entry, Text, Frame, Canvas, entre otros.

Pero, cómo se usan? Cómo se introduce un elemento en un Listbox? Cómo se le cambia el color de fondo y de letra a las cosas?

Label

Para el label se puede declarar una variable de tipo StringVar()

Ejemplo 13:

En este ejemplo, nos abre con raw_input un espacio para que escribamos algo. Luego simplemente imprimimos eso en un control de tipo Label, todo lo que tenemos que hacer es setear la variable de tipo StringVar(). De la siguiente manera:

```
from Tkinter import *

v0=Tk()
mitexto=StringVar()
label1=Label(v0,textvar=mitexto).pack()

escribir=raw_input("")
mitexto.set(escribir)

v0.mainloop()
```

```
v0,textvar=mitexto).pack()
_input("")
escribir)
```

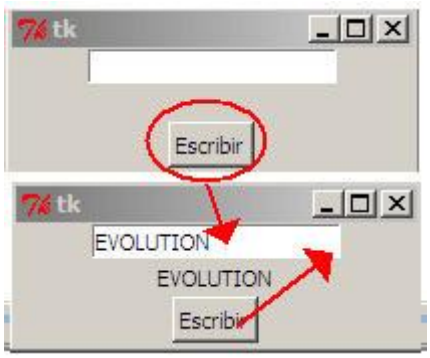
Como se puede ver en el ejemplo anterior, para darle valor a una variable del tipo StringVar() se escriba

variable.set(texto).

Ejemplo 14:

En el siguiente ejemplo veremos como se puede obtener valor de una variable de tipo StringVar(), utilizando variable.get()

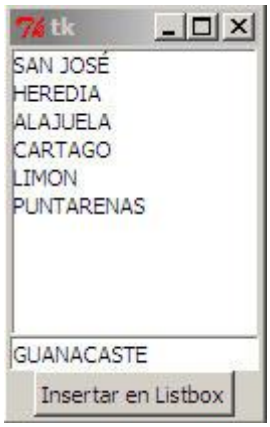
```
from Tkinter import *

v0=Tk()
v0.geometry("200x60")

def cambiar_stringvar(nuevotexto,stringvar):
    stringvar.set(nuevotexto)

mitexto=StringVar()
textoentry=StringVar()
entry1=Entry(v0,textvar=textoentry).pack()
label1=Label(v0,textvar=mitexto).pack()
b1=Button(v0,text="Escribir",command=lambda:
cambiar_stringvar(textoentry.get(),mitexto)).pack()

v0.mainloop()
```



Ejemplo 15:

Ahora veamos el objeto Listbox. Supongamos que queremos introducir una palabra en un control de tipo Listbox. Una forma de implementarlo seria:

```
from Tkinter import *

v0=Tk()

list1=Listbox(v0)
list1.pack()
mivalor=StringVar()
e1=Entry(v0,textvar=mivalor).pack()

def insertar_en_listbox():
    if mivalor.get() != "":
        list1.insert(END,mivalor.get())
    else: print "Por favor esciba algún texto"

b1=Button(v0,text="Insertar en
Listbox",command=insertar_en_listbox).pack()

v0.mainloop()
```



Si insertan muchos elementos se darán cuenta de que, en este tipo de controles normalmente aparece una Scrollbar automáticamente. El Scrollbar es otro tipo de control, y en Python, curiosamente, para que un Listbox tenga Scrollbar, deben unirse mediante un truco, por decirlo de alguna forma.

Ejemplo 16:

Para que el Listbox del ejemplo de arriba tenga un scrollbar, se debe hacer lo siguiente:

```
from Tkinter import *

v0=Tk()

def colocar_scrollbar(listbox,scrollbar):
    scrollbar.config(command=listbox.yview)
    listbox.config(yscrollcommand=scrollbar.set)
    scrollbar.pack(side=RIGHT, fill=Y)
    listbox.pack(side=LEFT, fill=Y)

frame1=Frame(v0)
frame1.pack()
scroll1=Scrollbar(frame1)
list1=Listbox(frame1)
list1.pack()
```

```
colocar_scrollbar(list1,scroll1)
mivalor=StringVar()
e1=Entry(v0,textvar=mivalor).pack()

def insertar_en_listbox():
    if mivalor.get() != "":
        list1.insert(END,mivalor.get())
    else: print "Por favor esciba algún texto"

b1=Button(v0,text="Insertar en
Listbox",command=insertar_en_listbox).pack()

v0.mainloop()
```

De esta forma al tener más elementos de los que permite visualizar, coloca automáticamente un Scrollbar.

Cabe destacar que en Scheme, el control Listbox traía su propio Scrollbar. Al menos eso recuerdo...

RESUMEN INTERFAZ GRÁFICA

TERCERA PARTE

3 Noviembre del 2010

Autor: Grupo 50, Segundo Semestre 2010 (Sede San Carlos)

10. PROPIEDADES DE LOS CONTROLES (Parte 2)

Ejemplo 17:

Ahora, suponiendo que tenemos una lista de cosas, y queremos que esa lista de cosas se inserten, una a una, en un listbox.
Para cosas como estas debemos implementar funciones , ya que de otra manera estaríamos repitiendo mucho código y
saturando nuestro proyecto, mientras que una función resume muchas instrucciones y lo hace más comprensible.

Esta es una solución para realizarlo:

```
from Tkinter import *

v0=Tk()

def colocar_scrollbar(listbox,scrollbar):
    scrollbar.config(command=listbox.yview)
    listbox.config(yscrollcommand=scrollbar.set)
    scrollbar.pack(side=RIGHT, fill=Y)
    listbox.pack(side=LEFT, fill=Y)

frame1=Frame(v0)
frame1.pack()
scroll1=Scrollbar(frame1)
list1=Listbox(frame1)
list1.pack()
colocar_scrollbar(list1,scroll1)

def cargarlistbox(lista,listbox):
    ind,largo=0,len(lista)
    while ind < largo:
```

```
        listbox.insert(END,lista[ind])
        ind+=1

ListaNombres=['Laura','Roger','Fabiola']
cargarlistbox(ListaNombres,list1)

v0.mainloop()
```

Ejemplo 18:

Este es un ejemplo de cómo imprimir el texto del elemento seleccionado en un Listbox, en un control del tipo Label. Esta es una de las formas con las que se puede implementar esto:

```
from Tkinter import *

v0=Tk()

def colocar_scrollbar(listbox,scrollbar):
    scrollbar.config(command=listbox.yview)
    listbox.config(yscrollcommand=scrollbar.set)
    scrollbar.pack(side=RIGHT, fill=Y)
    listbox.pack(side=LEFT, fill=Y)

frame1=Frame(v0)
frame1.pack()
scroll1=Scrollbar(frame1)
```

```
list1=Listbox(frame1)
list1.pack()
colocar_scrollbar(list1,scroll1)
mitexto=StringVar()
label1=Label(v0,textvar=mitexto)
label1.pack()

def cargarlistbox(lista,listbox):
    ind,largo=0,len(lista)
    while ind < largo:
        listbox.insert(END,lista[ind])
        ind+=1

ListaNombres=['Laura','Roger','Fabiola']

def imprimir_en_label():
    label1.after(100, imprimir_en_label) # Llamada recursiva con .after
    ind = list1.curselection()
    if list1.curselection() != ():
        sel = list1.get(ind)
        mitexto.set(sel)

cargarlistbox(ListaNombres,list1)
imprimir_en_label()

v0.mainloop()
```

La función imprimir_en_label es la que llama la atención en este caso, es una función que se llama así misma con el evento
objeto.after(tiempo(milisegundos),función).

Cada 100 milisegundos verifica cuál elemento está seleccionado.

Listbox.curselection() retorna la posición seleccionada, y
Listbox.get(posición) devuelve el texto seleccionado. Otra forma
de realizar esto es con una implementación de objeto.bind

Ejemplo 18:

Para este ejemplo lo que haremos es una función para limpiar un control del tipo Listbox. La función elimina todos los
elementos del listbox cuando uno le da click al botón ELIMINAR TODOS

```
from Tkinter import *

v0=Tk()

def colocar_scrollbar(listbox,scrollbar):
    scrollbar.config(command=listbox.yview)
    listbox.config(yscrollcommand=scrollbar.set)
    scrollbar.pack(side=RIGHT, fill=Y)
    listbox.pack(side=LEFT, fill=Y)

frame1=Frame(v0)
frame1.pack()
scroll1=Scrollbar(frame1)
list1=Listbox(frame1)
list1.pack()
colocar_scrollbar(list1,scroll1)
mitexto=StringVar()
label1=Label(v0,textvar=mitexto)
label1.pack()
```

```
b1=Button(v0,text="ELIMINAR TODOS",command=lambda:
limpiar_listbox(list1))
b1.pack()

def cargarlistbox(lista,listbox):
    ind,largo=0,len(lista)
    while ind < largo:
        listbox.insert(END,lista[ind])
        ind+=1

ListaNombres=['Laura','Roger','Fabiola','Yendry','Esteban','Marta','Elias
']

def imprimir_en_label():
    label1.after(100, imprimir_en_label) # Llamada recursiva con .after
    ind = list1.curselection()
    if list1.curselection() != ():
        sel = list1.get(ind)
        mitexto.set(sel)

def limpiar_listbox(listbox):
    while listbox.size() > 0:
        listbox.delete(0)

cargarlistbox(ListaNombres,list1)
imprimir_en_label()

v0.mainloop()
```

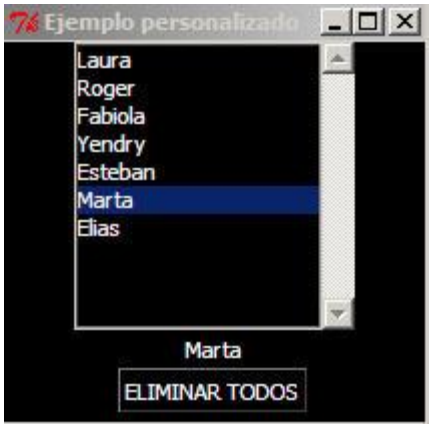
Ejemplo 19:

Para personalizar los colores de los objetos se utilizan:
bg: Se encarga del color de fondo
fg: Se encarga del color del texto

Hay otras propiedades más que se pueden encontrar en documentación sobre Tkinter por la web. Veamos el ejemplo de arriba
más personalizado

```
from Tkinter import *

v0=Tk()
v0.config(bg="black")
v0.title("Ejemplo personalizado")
v0.geometry("210x190")

def colocar_scrollbar(listbox,scrollbar):
    scrollbar.config(command=listbox.yview)
    listbox.config(yscrollcommand=scrollbar.set)
    scrollbar.pack(side=RIGHT, fill=Y)
    listbox.pack(side=LEFT, fill=Y)

frame1=Frame(v0,bg="black")
frame1.pack()
```

```
scroll1=Scrollbar(frame1)
list1=Listbox(frame1,bg="black",fg="white")
list1.pack()
colocar_scrollbar(list1,scroll1)
mitexto=StringVar()
label1=Label(v0,textvar=mitexto,bg="black",fg="white")
label1.pack()
b1=Button(v0,text="ELIMINAR TODOS",command=lambda:
limpiar_listbox(list1),bg="black",fg="white")
b1.pack()

def cargarlistbox(lista,listbox):
    ind,largo=0,len(lista)
    while ind < largo:
        listbox.insert(END,lista[ind])
        ind+=1

ListaNombres=['Laura','Roger','Fabiola','Yendry','Esteban','Marta','Elias
']

def imprimir_en_label():
    label1.after(100, imprimir_en_label) # Llamada recursiva con .after
    ind = list1.curselection()
    if list1.curselection() != ():
        sel = list1.get(ind)
        mitexto.set(sel)

def limpiar_listbox(listbox):
    while listbox.size() > 0:
        listbox.delete(0)

cargarlistbox(ListaNombres,list1)
imprimir_en_label()

v0.mainloop()
```



Existen otros elementos personalizables como

width: El ancho

height: El alto

border: Tamaño de borde

En interfaces más avanzadas se puede configurar el nivel de opacidad, logrando así el efecto de transparencia de las
ventanas más conocidas. Pero eso no es tan necesario mencionarlo

11. IMPRIMIENDO LA HORA CON TODO Y FECHA

Existe un módulo en Python llamado time. Utilizando la siguiente implementación se puede conseguir la hora:

Ejemplo 20:

```
import time

def imprimir_fecha():
    return str(time.localtime()[2]) + "/" + str(time.localtime()[1]) + "/" + str(time.localtime()[0]) + ", " + str(time.localtime()[3]) + ":" + str(time.localtime()[4]) + ":" + str(time.localtime()[5])

print imprimir_fecha()
```

Ejemplo 21:

Si uno deseara, podría conseguir imprimir más información, como los nombres del mes, entre otras cosas.

En el siguiente ejemplo hay una implementación de cómo se consigue imprimir la hora actual en un label.

```
import Tkinter
from Tkinter import *
import time

def imprimir_fecha():
    return str(time.localtime()[2]) + "/" + str(time.localtime()[1]) +
"/" + str(time.localtime()[0]) + ", " + str(time.localtime()[3]) + ":" +
str(time.localtime()[4]) + ":" + str(time.localtime()[5])

v0=Tk()
v0.title("Hora")
mifecha=StringVar()
l1=Label(v0,textvar=mifecha,font=(16))
l1.pack()
l2=Label(v0,text="",font=(16))
l2.pack()

def refresh_fecha():
    v0.after(1000,refresh_fecha)
    mifecha.set(imprimir_fecha())
    l2.config(text=imprimir_fecha())

refresh_fecha()
v0.mainloop()
```

Además en este ejemplo podemos ver que existen dos formas de cambiar el texto de un control Label, utilizando
objeto.config(text='loquesea') o creando una variable de tipo StringVar() y modificándola directamente.

El ejemplo sirve incluso para crear efectos animaciones, y textos cambiantes. Eso queda a imaginación del programador.

12. GUARDANDO Y ABRIENDO UN ARCHIVO CON TEXTO

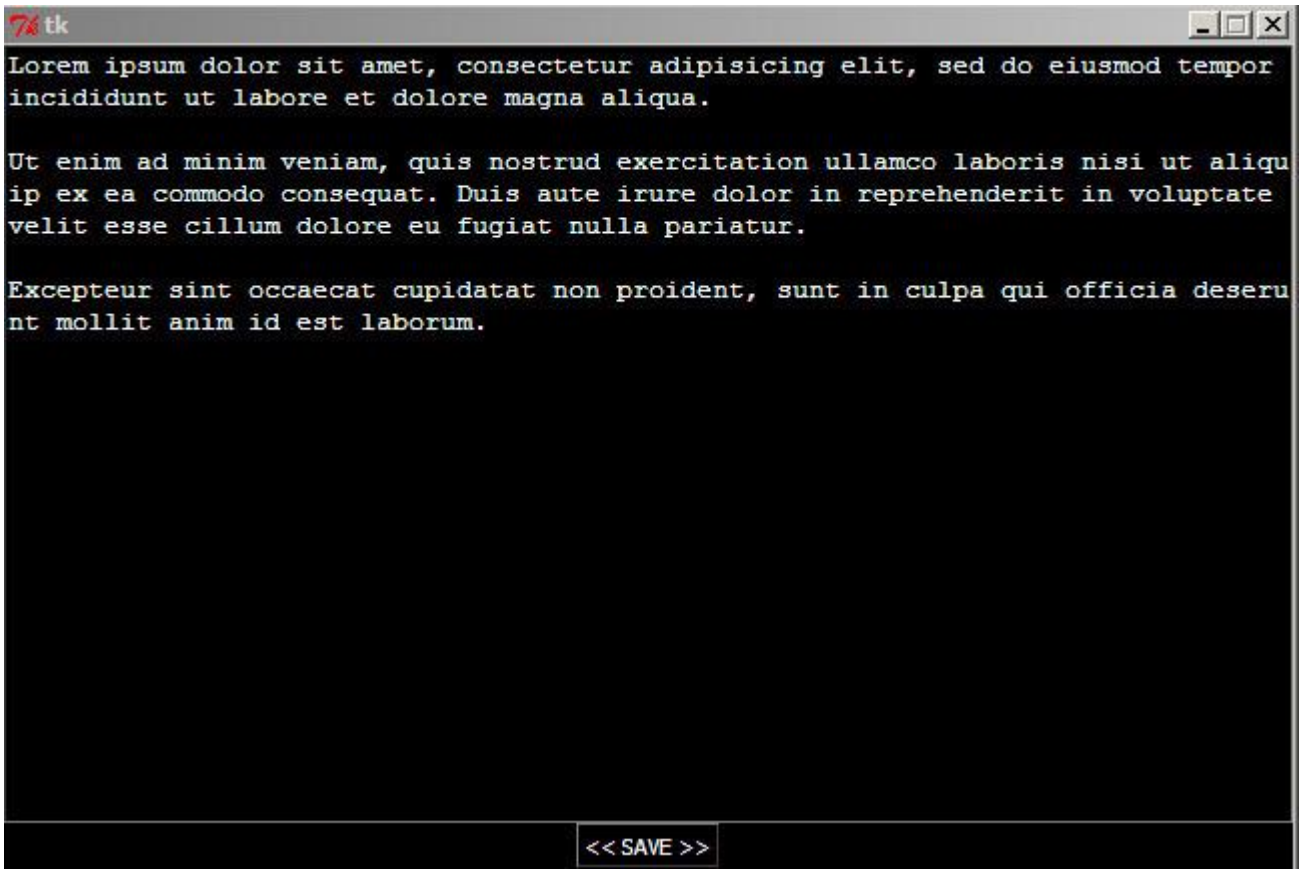
Ahora vamos a hacer uso de varios módulos. Esta aplicación abre una ventana con un control Text y un control Button.

Ejemplo 22:

El siguiente código guarda un archivo de texto en una ruta específicada. Y luego, si cerramos el programa, cuando lo
volvemos a abrir, nos devuelve el texto que guardamos anteriormente (Puedes cambiar el enlace si estás trabajando en
Ubuntu u otro Sistema Operativo que no lee disco C como en Windows)

```
# -*- coding: utf-8 -*-

import Tkinter
from Tkinter import *
import codecs
import os

v0 = Tkinter.Tk()
v0.config(bg="black")
v0.resizable(0,0)

def doit(f): v0.after(100, f)
def imprimir(textcontrol): print textcontrol.get('1.0', END+'-1c')
```

```
def escribir_en_archivo(enlace):
    f = codecs.open(enlace,"w","utf-8")
    texto = t1.get('1.0', END+'-1c')
    f.write(texto)
    f.close()

def abrir_archivo(enlace):
    if os.path.exists(enlace):
        f = open(enlace,"r")
        h= f.read()
        t1.insert(END,h)
        f.close()

t1=Text(v0)
t1.config(bg="black",fg="white")
t1.pack()

b1 = Button(v0,text="<< SAVE >>",command=lambda:
doit(escribir_en_archivo('C:\hola.txt')))
b1.config(bg="black",fg="white")
b1.pack()

abrir_archivo('C:\hola.txt')
v0.mainloop()
```



Este permite guardar y cargar información en archivos de texto, siempre y cuando sea string, lo guardará adecuadamente.

Cabe destacar que este pequeño programa tiene un código un tanto simple, pero tiene algunos detalles interesantes. Por ejemplo, no es igual utilizar controles Entry (una sola línea) que controles Text (Varias líneas de texto)

Su información es diferente, sus métodos, y algunas cosas para leer y escribir información en un control de tipo Text, es diferente a la manipulación del objeto Entry. Además en este ejemplo ni siquiera podemos ver toda la capacidad que tiene el objeto Text. Es uno de los objetos más potentes que tiene la interfaz Tkinter.

El segundo detalle a tomar en cuenta, es que, aunque parezca fácil, no es para nosotros obvio a simple vista que hay algo
en este documento, lo que e conoce como codificación.

La codificación es para que no se nos caiga nuestra aplicación cuando intentemos guardar en un archivo, caracteres tales
como tildes, eñes, y otros caracteres que se necesitan. Para ello incluso se importa un módulo llamado codecs, y se
utiliza utf-8, que les recomiendo realicen una búsqueda en google y revisen la información sobre la diferente codificación
de caracteres.

Otro detalle raro es la forma en la que se consigue el texto de un control del tipo Text

```
'1.0', END+'-1c'
```

Cuando en los controles Entry basta con Entry.get(mi_variable_entry)

Finalmente, para cargar el archivo primero se verifica si el archivo existe, esto lo podemos realizar con el módulo os, con
el que podemos verificar si un archivo existe en el sistema, si existe entonces lo abrirá.

13. GUARDANDO Y ABRIENDO LISTAS CON PICKLE

El módulo Pickle permite guardar las listas como estructuras de datos y luego recuperarlas desde el archivo. Esto nos
permitiría guardar toda la información de una lista y recuperarla tal cual.

Ejemplo 23:

En este ejemplo importamos el módulo pickle y creamos unas funciones para guardar y abrir listas.

```
import pickle

def guardar_lista(lista,ruta):
    fichero = file(ruta, "w")
    nl=lista
    pickle.dump(nl, fichero)

def cargar_lista(ruta):
    fichero = file(ruta)
    lista_recuperada = pickle.load(fichero)
    return lista_recuperada

guardar_lista([1,2,3,4,5,6,7],"datos.txt")
print cargar_lista("datos.txt")
```

En este ejemplo guardamos una lista con los números del 1 al 7 en el archivo datos.txt que se creará en la misma carpeta
donde está guardado el archivo.py donde se encuentra el código de arriba, pero nosotros podemos cambiar la ruta, por
ejemplo 'C:\datos.txt'

RESUMEN INTERFAZ GRÁFICA

CUARTA PARTE

4 Noviembre del 2010

Autor: Grupo 50, Segundo Semestre 2010 (Sede San Carlos)

14. GUARDANDO Y CARGANDO OBJETOS CON PICKLE

El módulo pickle del que hablábamos en la parte 3 del tutorial, no solo sirve para guardar listas, si no para cualquier
tipo de dato en general. Hasta objetos.

Ejemplo 24:

A continuación se presenta una forma de cómo guardar objetos en Python:

```
import pickle

def guardar_lista(lista,ruta):
    fichero = file(ruta, "w")
    nl=lista
    pickle.dump(nl, fichero)

def cargar_lista(ruta):
    fichero = file(ruta)
    lista_recuperada = pickle.load(fichero)
    return lista_recuperada

class persona:
    def __init__(self,nombre):
        self.nombre=nombre
        print "Ha nacido",nombre

Roxana=persona("Roxana")

ListaPersonas=[Roxana]
print "Esta es ListaPersonas:",ListaPersonas

guardar_lista(ListaPersonas,"C:\salvaobjetos.txt")
ListaPersonas2 = cargar_lista("C:\salvaobjetos.txt")
print "Esta es ListaPersonas2",ListaPersonas2

print "Nombre de primer elemento en
ListaPersonas:",ListaPersonas[0].nombre
print "Nombre de primer elemento en
ListaPersonas2:",ListaPersonas2[0].nombre
```

En este ejemplo se utilizan las funciones para guardar listas. Se guarda directamente la lista con los objetos del tipo
persona, clase que ha sido creada para este ejemplo y que únicamente contiene el atributo nombre.

En este ejemplo se realiza lo siguiente:

1. Se crea el objeto persona que contiene un atributo nombre
2. Se crea una variable Roxana de tipo persona, cuyo atributo es 'Roxana'
3. Se crea una lista llamada ListaPersonas, que contiene la variable Roxana
4. Se salva ListaPersonas en el archivo 'C:\salvaobjetos.txt'
5. Se crea una lista llamada ListaPersonas2, que contiene lo mismo que el archivo 'C:\salvaobjetos.txt'
6. Se imprime cuidadosamente cada paso en el código. Por lo que queda comprobado que se guardan también los objetos con el módulo pickle

15. COLOCANDO UNA IMÁGEN GIF ESTÁTICA CON UN LABEL

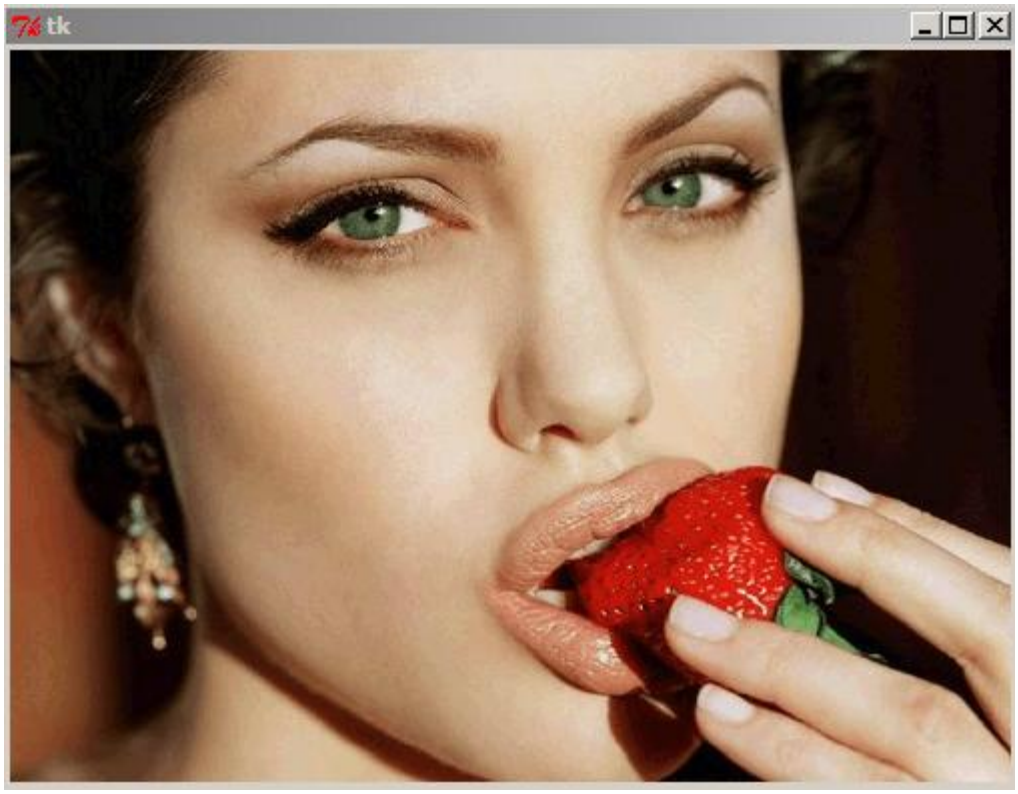
Las imágenes existen varios formatos, es algo muy básico que ya deberíamos saber. Existen formatos como JPG, PNG, BMP, GIF, entre otros.

Ejemplo 25:

Una forma de colocar una imágen GIF en una ventana sería la siguiente:

```
from Tkinter import *

v0=Tk()
imagen1=PhotoImage(file="C:\COUNTER.gif")
label1 = Label(v0, image=imagen1)
label1.grid(row=1,column=1)

v0.mainloop()
```

Antes de que comiencen a probar todos estos ejemplos, pueden bajarse la foto de Angelina de aquí:
http://i.imgur.com/qTKZv.gif
Es una imágen que ya tiene el formato GIF correcto y no presentará problemas a la hora de cargarlo.



Cabe señalar que para cambiarle el formato a una imágen basta con quitarle la extensión que tiene y ponerle otra. Debe ser previamente abierta con un programa que permita abrir el formato en el que está la imágen original, y guardarla en formato GIF.

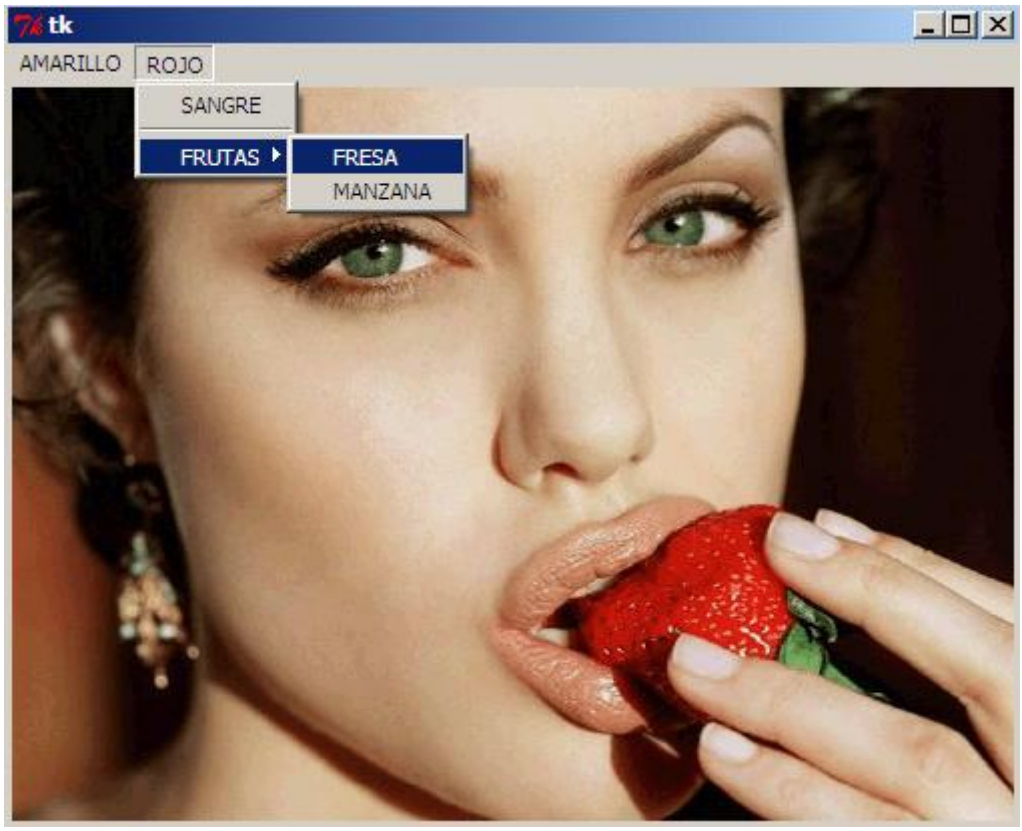
16. CREANDO MENÚES EN LA PARTE SUPERIOR DE LA VENTANA

Ejemplo 26:

Vamos a agregarle a la hermosa Angelina Jolie unos cuántos menúes arriba. Supongamos que vamos a dividir cosas en submenúes según su color

```
from Tkinter import *
```

```
v0=Tk()
imagen1=PhotoImage(file="C:\COUNTER.gif")
label1 = Label(v0, image=imagen1)
label1.grid(row=1,column=1)

def imprimir(texto): print texto

menu1 = Menu(v0)
v0.config(menu=menu1)
menu1_1 = Menu(menu1, tearoff=0)
menu1.add_cascade(label="AMARILLO", menu=menu1_1)
menu1_1_1 = Menu(menu1_1, tearoff=0)
menu1_1.add_cascade(label="FRUTAS", menu=menu1_1_1)
menu1_1_1.add_command(label="BANANO",command=lambda: imprimir("BANANO"))
menu1_1_1.add_command(label="PIÑA",command=lambda: imprimir("PIÑA"))

menu1_2 = Menu(menu1, tearoff=0)
menu1.add_cascade(label="ROJO", menu=menu1_2)
menu1_2.add_command(label="SANGRE",command=lambda: imprimir("SANGRE"))
menu1_2.add_separator()
menu1_2_1 = Menu(menu1_2, tearoff=0)
menu1_2.add_cascade(label="FRUTAS", menu=menu1_2_1)
menu1_2_1.add_command(label="FRESA",command=lambda: imprimir("FRESA"))
menu1_2_1.add_command(label="MANZANA",command=lambda:
imprimir("MANZANA"))
v0.mainloop()
```



Cada menú que contiene elementos se crea, y es a ese menú al que se introduce en un menú padre. Se le pueden agregar separadores, comandos clickeables, cascadas, entre otras cosas.

Además el menú padre hay que asociarlo a la ventana directamente con ventanaprincipal.config(menu=menupadre), tal como se muestra arriba

17. CREANDO NUESTRA SPLASH SCREEN

Una Splash Screen es una ventana que nos aparece cuando abrimos un programa. Por ejemplo lo siguiente es una splash screen:

Ejemplo 27:

Para crear nuestro Splash Screen, lo que debemos hacer es crear una pausa entre el Splash Screen, y la ventana principal,
la cual debe estar oculta. Eso lo podemos realizar de la siguiente manera:

```
from Tkinter import *

v0=Tk()
imagen1=PhotoImage(file="C:\COUNTER.gif")
label1 = Label(v0, image=imagen1)
label1.grid(row=1,column=1)

def imprimir(texto): print texto
def mostrar(ventana): return ventana.deiconify # Muestra una ventana
def ocultar(ventana): return ventana.withdraw() # Oculta una ventana
def ejecutar(f): v0.after(100, f)

menu1 = Menu(v0)
v0.config(menu=menu1)
menu1_1 = Menu(menu1, tearoff=0)
```

```
menu1.add_cascade(label="AMARILLO", menu=menu1_1)
menu1_1_1 = Menu(menu1_1, tearoff=0)
menu1_1.add_cascade(label="FRUTAS", menu=menu1_1_1)
menu1_1_1.add_command(label="BANANO",command=lambda: imprimir("BANANO"))
menu1_1_1.add_command(label="PIÑA",command=lambda: imprimir("PIÑA"))

menu1_2 = Menu(menu1, tearoff=0)
menu1.add_cascade(label="ROJO", menu=menu1_2)
menu1_2.add_command(label="SANGRE",command=lambda: imprimir("SANGRE"))
menu1_2.add_separator()
menu1_2_1 = Menu(menu1_2, tearoff=0)
menu1_2.add_cascade(label="FRUTAS", menu=menu1_2_1)
menu1_2_1.add_command(label="FRESA",command=lambda: imprimir("FRESA"))
menu1_2_1.add_command(label="MANZANA",command=lambda:
imprimir("MANZANA"))

v1=Toplevel(v0)
v1.geometry("400x200")
v1.config(bg="black")
v1.protocol("WM_DELETE_WINDOW", "onexit")
v1.resizable(0,0)

ocultar(v0)
def cerrar_splashscreen():
    ejecutar(ocultar(v1))
    ejecutar(mostrar(v0))
v1.after(4000,cerrar_splashscreen)
Label(v1,text="BIENVENIDO A NUESTRA
APLICACIÓN",bg="black",fg="white",font=(15)).pack()
Label(v1,text="BIENVENIDO A NUESTRA
APLICACIÓN",bg="black",fg="white",font=(15)).pack()
Label(v1,text="BIENVENIDO A NUESTRA
APLICACIÓN",bg="black",fg="white",font=(15)).pack()
Label(v1,text="BIENVENIDO A NUESTRA
APLICACIÓN",bg="black",fg="white",font=(15)).pack()
```

```
Label(v1,text="BIENVENIDO A NUESTRA
APLICACIÓN",bg="black",fg="white",font=(15)).pack()

v0.mainloop()
```

Utiliza el método after para provocar un efecto de espera, y luego oculta una ventana y muestra la otra.

18. ACOMODANDO LOS ELEMENTOS: TOP, BOTTOM, LEFT, RIGHT

Una de las cosas que uno puede tardar más haciendo con Python, es acomodando los elementos. Si estuviésemos programando en Java con Netbeans, sería más cómodo pues disponemos de un panel como el siguiente:

Por lo que para nosotros que debemos hacerlo a pura línea de código, puede ser un poco más complicado. Sin embargo, como es para propósito meramente de aprendizaje, es indispensable haber hecho aunque sea una vez en toda tu carrera (Que habrá más) el trabajo de Interfaz Gráfica solo utilizando código, sin herramientas externas.

Para acomodar los elementos en Python, al igual que en Scheme, y me imagino que en la mayoría de lenguajes de programación que posean un módulo de Interfaz Gráfica que se haga llamar standard, se utilizan paneles, y dos propiedades de los objetos gráficos, llamados .pack() y .grid()

Ejemplo 28:

Las ventanas tienen arriba, abajo, izquierda y derecha. En inglés se llaman TOP, BOTTOM, LEFT, RIGHT. Podemos hacer uso de esto para acomodar los elmentos a un lado o al otro de la ventana, de la siguiente forma:

```
from Tkinter import *

v0=Tk()
v0.geometry("200x200")

b1=Button(v0,text="ARRIBA").pack(side=TOP)
b3=Button(v0,text="ABAJO").pack(side=BOTTOM)
b2=Button(v0,text="IZQUIERDO").pack(side=LEFT)
b4=Button(v0,text="DERECHO").pack(side=RIGHT)

v0.mainloop()
```

Ejemplo 29:

Y el orden de colocación en el código sí importa. Nóte que primero está TOP, luego BOTTOM, luego LEFT, y luego RIGHT. Pero si por ejemplo colocamos este orden

```
from Tkinter import *

v0=Tk()
v0.geometry("200x200")
```

```
b1=Button(v0,text="ARRIBA").pack(side=TOP)
b2=Button(v0,text="IZQUIERDO").pack(side=LEFT)
b3=Button(v0,text="ABAJO").pack(side=BOTTOM)
b4=Button(v0,text="DERECHO").pack(side=RIGHT)

v0.mainloop()
```

Tendremos que los botones no se alinean de manera correcta.

Ejemplo 30:

Es importante recordar que también podemos colocar el código de la siguiente forma:

```
from Tkinter import *

v0=Tk()
v0.geometry("200x200")

b1=Button(v0,text="ARRIBA")
b1.pack(side=TOP)
b3=Button(v0,text="ABAJO")
b3.pack(side=BOTTOM)
b2=Button(v0,text="IZQUIERDO")
b2.pack(side=LEFT)
b4=Button(v0,text="DERECHO")
```

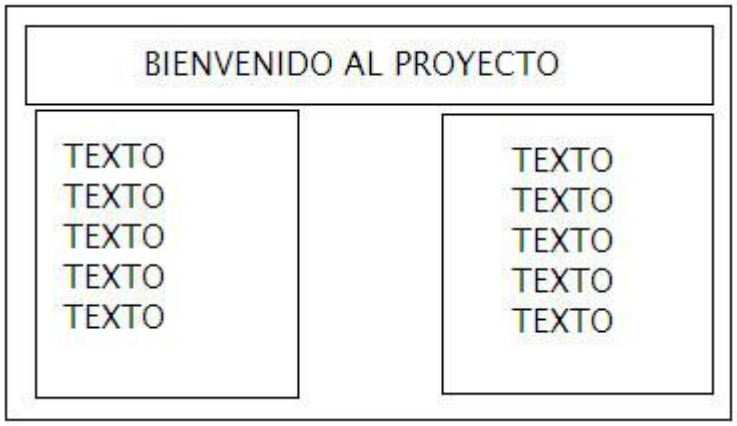
```
b4.pack(side=RIGHT)

v0.mainloop()
```

19. ACOMODANDO LOS ELEMENTOS: FRAMES

Una de las cosas a las que lleva tiempo acostumbrarse es a los Frames. Si no se tiene cuidado o se usan con exceso, un proyecto puede contener más frames que cualquier otro tipo de elemento gráfico.

Ejemplo 31:

Un frame es un panel que sirve como padre para otros elementos en una ventana. Casi siempre que uno va a trabajar con frames en un proyecto, es recomendable primero realizar un dibujo en papel o con algún editor gráfico, y pensar en cómo quedará al final. Acá tenemos un ejemplo de cómo se pueden acomodar los elementos con Frame




```
from Tkinter import *

v0=Tk()
v0.geometry("200x200")
```

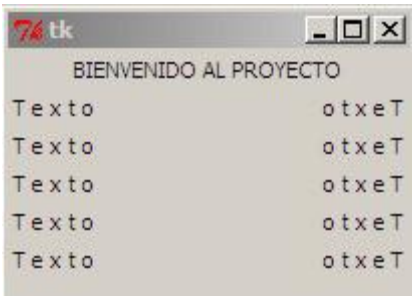
```
l1=Label(v0,text="BIENVENIDO AL PROYECTO")
l1.pack()

frame0=Frame(v0)
frame0.pack(side=TOP)

frame1=Frame(frame0)
Label(frame1,text="T e x t o").pack()
Label(frame1,text="T e x t o").pack()
Label(frame1,text="T e x t o").pack()
Label(frame1,text="T e x t o").pack()
Label(frame1,text="T e x t o").pack()
frame1.pack(side=LEFT)

frame2=Frame(frame0)
Label(frame2,text="o t x e T").pack()
Label(frame2,text="o t x e T").pack()
Label(frame2,text="o t x e T").pack()
Label(frame2,text="o t x e T").pack()
Label(frame2,text="o t x e T").pack()
frame2.pack(side=RIGHT)

v0.mainloop()
```

Ejemplo 32

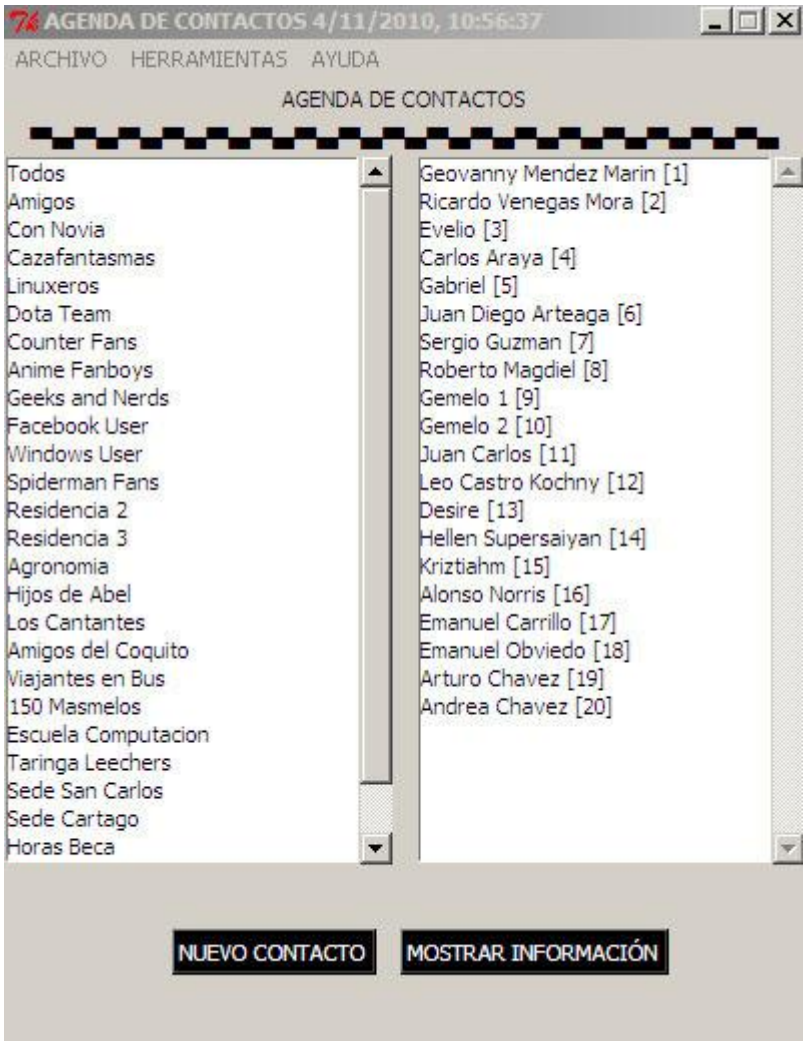
También tenemos propiedades para algunos objetos gráficos, que sirven para acomodamiento en las ventanas. Se llaman expand y fill. Veamos un ejemplo de cómo funciona fill

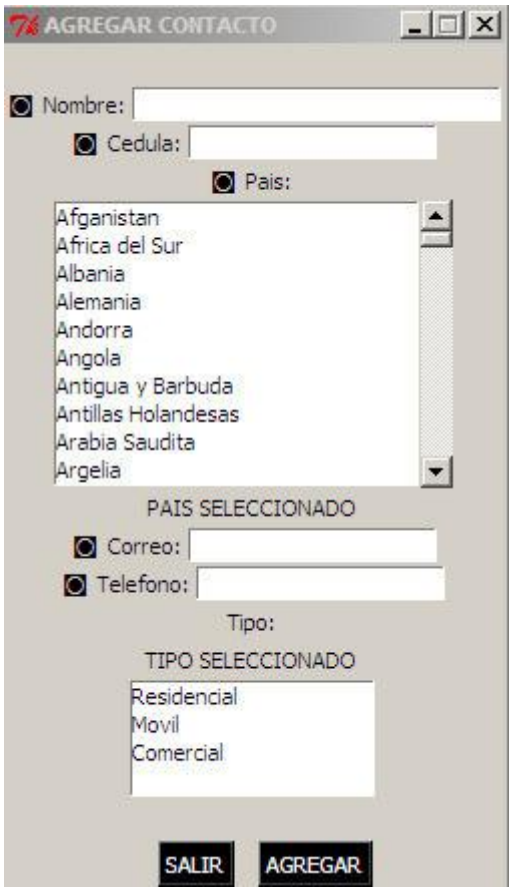
```
from Tkinter import *
```

```
v0=Tk()
v0.geometry("200x200")

l1=Label(v0,text="BIENVENIDO AL PROYECTO")
l1.pack()

frame0=Frame(v0)
frame0.pack(side=TOP,fill=BOTH)

frame1=Frame(frame0)
Label(frame1,text="T e x t o").pack()
Label(frame1,text="T e x t o").pack()
Label(frame1,text="T e x t o").pack()
Label(frame1,text="T e x t o").pack()
Label(frame1,text="T e x t o").pack()
frame1.pack(side=LEFT)

frame2=Frame(frame0)
Label(frame2,text="o t x e T").pack()
Label(frame2,text="o t x e T").pack()
Label(frame2,text="o t x e T").pack()
Label(frame2,text="o t x e T").pack()
Label(frame2,text="o t x e T").pack()
frame2.pack(side=RIGHT)

v0.mainloop()
```



En la documentación viene más información sobre expand y fill. Son conceptos sobre el ordenamiento de los objetos a los que hemos de acostumbrarnos para que al final le demos un acabado a la interfaz que se adapte a nuestras necesidades. Aquí hay unas ventanas acomodadas utilizando el método de Frames

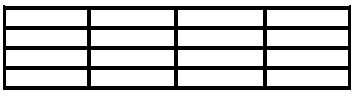

AGENDA DE CONTACTOS 4/11/2010, 10:56:37
ARCHIVO HERRAMIENTAS AYUDA
AGENDA DE CONTACTOS
Todos
Amigos
Con Novia
Cazafantasmas
Linuxeros
Dota Team
Counter Fans
Anime Fanboys
Geeks and Nerds
Facebook User
Windows User
Spiderman Fans
Residencia 2
Residencia 3
Agronomia
Hijos de Abel
Los Cantantes
Amigos del Coquito
Viajantes en Bus
150 Masmelos
Escuela Computacion
Taringa Leechers
Sede San Carlos
Sede Cartago
Horas Beca
Geovanny Mendez Marin [1]
Ricardo Venegas Mora [2]
Evelio [3]
Carlos Araya [4]
Gabriel [5]
Juan Diego Arteaga [6]
Sergio Guzman [7]
Roberto Magdiel [8]
Gemelo 1 [9]
Gemelo 2 [10]
Juan Carlos [11]
Leo Castro Kochny [12]
Desire [13]
Hellen Supersaiyan [14]
Kriztiahm [15]
Alonso Norris [16]
Emanuel Carrillo [17]
Emanuel Obviedo [18]
Arturo Chavez [19]
Andrea Chavez [20]
NUEVO CONTACTO
MOSTRAR INFORMACIÓN



20. ACOMODANDO LOS ELEMENTOS: GRID

Otra forma de acomodar los elementos en una ventana es con el método grid. Anteriormente vimos que el método .pack tenía atributos como side, fill y expand.

El método grid tiene como atributos row y column. En una matriz, row (en español fila) son las líneas horizontales (conocidas también como vectores), y column (columnas..), son las líneas verticales.



Esta es la forma de comprender el método grid, utiliza la ventana como si fuera una matriz, y ordena los elementos como entradas de esa matriz. Veamos el ejemplo con botones:

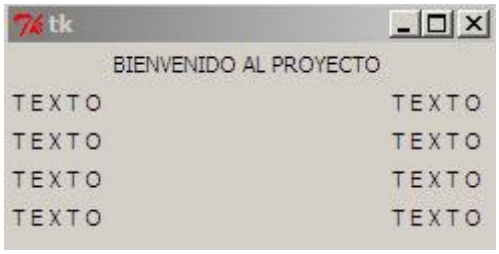
```
from Tkinter import *
```

```
v0=Tk()
v0.geometry("200x200")

Button(v0,text="A").grid(row=1,column=1)
Button(v0,text="B").grid(row=1,column=2)
Button(v0,text="C").grid(row=1,column=3)
Button(v0,text="D").grid(row=1,column=4)
Button(v0,text="E").grid(row=2,column=1)
Button(v0,text="F").grid(row=2,column=2)
Button(v0,text="G").grid(row=2,column=3)
Button(v0,text="H").grid(row=2,column=4)
Button(v0,text="I").grid(row=3,column=1)
Button(v0,text="J").grid(row=3,column=2)
Button(v0,text="K").grid(row=3,column=3)
Button(v0,text="L").grid(row=3,column=4)
Button(v0,text="M").grid(row=4,column=1)
Button(v0,text="N").grid(row=4,column=2)
Button(v0,text="Ñ").grid(row=4,column=3)
Button(v0,text="O").grid(row=4,column=4)

v0.mainloop()
```

Y el ejemplo de más arriba pero con grid, sería de la siguiente forma:

```
from Tkinter import *

v0=Tk()
v0.geometry("242x100")

Label(v0,text="").grid(row=1,column=1)
Label(v0,text="BIENVENIDO AL PROYECTO").grid(row=1,column=2)
Label(v0,text="").grid(row=1,column=3)
Label(v0,text="T E X T O").grid(row=2,column=1)
Label(v0,text="T E X T O").grid(row=2,column=3)
Label(v0,text="T E X T O").grid(row=3,column=1)
Label(v0,text="T E X T O").grid(row=3,column=3)
```

```
Label(v0,text="T E X T O").grid(row=4,column=1)
Label(v0,text="T E X T O").grid(row=4,column=3)
Label(v0,text="T E X T O").grid(row=5,column=1)
Label(v0,text="T E X T O").grid(row=5,column=3)
Label(v0,text="T E X T O").grid(row=6,column=1)
Label(v0,text="T E X T O").grid(row=6,column=3)

v0.mainloop()
```



Dos observaciones importantes. La primera, no mezcles ObjetoBoton.pack() y ObjetoBoton.grid() en una misma ventana. Y la segunda, Es que podrías introducir los elementos en frames, y luego utilizar el grid para acomodar los Frames. Se debe planear la manera en cómo irán acomodados los elementos.

Además una de las cosas que debemos tomar en cuenta cuando programamos una interfaz, es que si va a cambiar el tamaño de la ventana, debemos configurarla adecuadamente para que se mantenga la proporción del aspecto. Por que si maximiza la ventana, los elementos se van a ver mal. Una solución para esto es prohibir que se pueda cambiar el tamaño de la ventana, o buscar maneras para que los elementos 'se mantengan en su sitio' aún cuando la ventana cambie de tamaño

Finalmente, el grid puede ser a veces engorroso y dar resultados no deseados, si no sabemos bien como configurar bien su aspecto. Por eso lo mejor es mezclarlo con Frames y darle el acabado deseado.

Para una mejor documentación de grid y demás elementos de la interfaz Tkinter en general, diríjase al siguiente link:
http://effbot.org/tkinterbook/grid.htm