Guia Tkinter Documentation

Publicación 0.1.1

Alvarez Alejandro

31 de December de 2015

Algo muy buscado en Python son las guías sobre interfaces gráficas ya sea PyGTK, PyQT, WxPython o Tkinter entre las mas conocidas teniendo cada una sus ventajas y desventajas, así como cada una tiene facilidades y complicaciones en su uso y aplicación. En esta oportunidad les daremos un espacio a Tkinter e intentaremos ser lo mas lo mas claro posible sin pasarnos por alto los pequeños detalles en lo que compone la creación de una interfaz gráfica en Python con el toolkit Tkinter.

Contenido:

CAPÍTULO 1

Introduccion

Version 0.1.1. (Descarga)

Figura 1.1: Branch develop

Figura 1.2: Branch master

Puede descargar la versión más reciente de esta guía gratuitamente en la web https://github.com/eliluminado/tutorial_tkinter

1.1 Agradecimientos

Ante todo gracias a todos los lectores y a aquellos que me enviaron correos con consultas y sugerencias y desde ya a los ayudaron a mejorar esta guía

CAPÍTULO 2

Creditos

Esta guia es escrita y mantenida por Alejandro Alvarez <elilluminado@codigopython.com.ar>

2.1 Colaboradores

Ninguno todavia. Por que no ser el primero?

CAPÍTULO 3

LICENCIA

CAPÍTULO 4

Historial de cambios

4.1 Version 0.1.2 (En desarrollo)

4.1.1 Añadidos

- Se especifico argumentos, valores de retorno y excepciones para los metodos de Tkinter
- Se subio guia a Readthedocs
- Validacion con tox y Travis CI

4.1.2 Correcciones

- Se mejoro salida al usar Sphinx
- Se removieron metodos y opciones que no estaban disponibles en tkinter pero si en tk
- Se mejoraron las capturas de pantalla para los ejemplos
- El codigo mostrado es compatible con Python 2.x y 3.x
- Se actualizo la licencia a la version 4.0

4.2 Version 0.1.1

- Se retoma el desarrollo de la guia utilizado Sphinx como motor

CAPÍTULO 5

Introduccion

Sean bienvenidos a esta noble guía que pretende a lo largo de sus capítulos mostrar las herramientas y conocimientos necesarios para el desarrollo de interfaces gráficas con las librerías Tk y el lenguaje de programación Python. El contenido de esta guía no pretende ser una referencia completa de lo que puede lograr pero si servir como una base al poco contenido que se logra encontrar en español sobre Tkinter.

CAPÍTULO 6

Acerca de Tk

Tkinter es un binding de la biblioteca gráfica Tcl/Tk para el lenguaje de programación Python, con estos queremos decir que Tk se encuentra disponible para varios lenguajes de programación entre los cuales se encuentra Python con el nombre de Tkinter. Este no es mas que una capa de esta librería para el lenguaje Python con lo cual usar Tk en otro lenguaje no nos supondrá un inconveniente.

Se considera un estándar para la interfaz gráfica de usuario (GUI) para Python y es el que viene por defecto con la instalación para Microsoft Windows y preinstalado en la mayoría de las distribuciones de GNU/Linux. Con Tkinter podremos conseguir resultados casi tan buenos como con otras librerías gráficas siempre teniendo en cuenta que quizás con otras herramientas podamos realizar trabajos mas complejos donde necesitemos una plataforma mas robusta, pero como herramienta didáctica e interfaces sencillas nos sobrara, dándonos una perspectiva de lo que se trata el desarrollo de una parte muy importante de una aplicación si deseamos distribuirla. Gracias a Tkinter veremos como interactuar con el usuario pidiéndole el ingreso de datos, capturando la pulsación de teclas, movimientos del mouse, entre algunas de las cosas que podremos lograr.

6.1 Historia

6.2 Virtudes y limitaciones

CAPÍTULO 7

Que es una interfaz grafica

7.1 Buenas practicas

7.1.1 Colores

En esta parte nos detendremos un momento para analizar la utilizacion de colores en nuestra aplicacion y la utilizacion de paletas y esquemas de colores que nos permitan conseguir aspectos mas prolijos y agradables a la vista.

CAPÍTULO 8

Instalando Tkinter

8.1 Linux

8.2 Windows

8.3 Mac

CAPÍTULO 9

Sobre esta guía

Para los que deseen colaborar con esta guía así como en su contenido, pueden hacerlo enviandome un correo electronico (eliluminado@codigopython.com.ar), de preferencia en español, o a traves de GitHub desde el canal para envio de fallos o correcciones desde la siguiente url: https://github.com/eliluminado/tutorial_tkinter/issues

Para los mas experimentados pueden colaborar realizando un fork de la guia y luego enviar los cambios entrando en https://github.com/eliluminado/tutorial_tkinter

9.1 Workflow

Por hacer

Ampliar contenido

El desarrollo y las correcciones de la guia se realizan sobre la rama *develop* y se mantiene la rama *master* unicamente para el contenido que se considere corregido y correcto.

Como recomendacion se agradece chequear que Sphinx no tire errores durante la compilacion, para esto esta tox.

Por hacer

Ampliar contenido sobre uso de tox

Tambien pueden asegurarse de hacerlo antes de hacer un commit modificando o generando el archivo `.git/hooks/pre-commit` y reemplazar el contenido de este por la siguiente linea

```
#!/bin/sh

tox
```

Y para el fichero `.git/hooks/pre-push` hacemos lo mismo pero con el siguiente codigo.

```
#!/bin/sh

tox -e linkcheck
```

Esta recomendacion lo que hace es que despues de que guardes tus cambios con Git te asegures de que Sphinx lo interpreta correctamente y el segundo fichero mostrado evita que subas cambios a GitHub donde hayan links rotos.

9.2 Estilos

Las imagenes como captura de los ejemplos mostrados, deben ser preferentemente en formato PNG.

Cuando se coloquen imagenes siempre usar la opcion *:alt:* de reStructuredText.

Cuando escriban que los renglones no superen los 120 caracteres.

9.3 Traducciones

Por hacer

Ampliar contenido

9.4 Referencias

Estos son los recursos web que se tuvieron en cuenta para la realizacion de esta guia (ademas de la documentacion oficial de la libreria tk).

9.4.1 Aun no revisadas

http://zetcode.com/gui/tkinter/

http://modcopy.sourceforge.net/tips.html

http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/index.html

http://tkinter.unpythonic.net/wiki/

http://staff.washington.edu/rowen/ROTKFolklore.html

http://staff.washington.edu/rowen/TkinterSummary.html

http://www.ferg.org/thinking_in_tkinter/all_programs.html

http://effbot.org/tkinterbook/

http://www.tkdocs.com/

CAPÍTULO 10

Empezando por lo básico

Lo primero que debemos hacer al igual que con otros módulos, es que debemos importarlo para poder comenzar a utilizarlo, y al igual que con otros módulos no hay una sola forma de hacerlo.

La primer forma (y la mas popular):

Python 2.x

```
    from Tkinter import *
```

Python 3.x

```
    from tkinter import *
```

Y la segunda:

Python 2.x

```
    import Tkinter
```

Python 3.x

```
    import tkinter
```

Nota: En las versiones 3.x de Python el modulo Tk se debe llamar de esta forma *"from tkinter import *"* y no de esta otra forma *"from Tkinter import *"*, notar la t minuscula en el nombre, es un cambio menor que hay que tener en cuenta si están trabajando con Python 3.

La diferencia entre usar la primera o la segunda forma es la misma con la que nos podemos encontrar a la hora de importar un modulo en Python, para verlo les muestro esta diferencia utilizando al modulo *'time'* y usamos el siguiente ejemplo:

```
1 import time
2 time.sleep(10)
```

```
1 from time import sleep
2 sleep(10)
```

Usando la forma *"import Tkinter"* cada vez que utilicemos una función de este modulo tendremos que anteponer la palabra *'Tkinter'*, en cambio usando la segunda forma *"from Tkinter import *"* simplemente deberemos usar el nombre de la función sin el nombre del modulo. Hay otras formas pero dependerá de tu forma de trabajar o la que te resulte mas cómoda, en todo caso puedes darle un vistazo a esta traducción de la guía de estilo escrita por Guido van Rossum y Barry Warsaw http://mundogeek.net/traducciones/guia-estilo-python.htm

Por ahora a modo didactico usaremos una variacion de la segunda forma para que los ejemplos sean mas claros, pero esto queda a tu eleccion. Ahora si retomemos nuestro camino.

Una observación que tendremos que tener antes de continuar es no debemos pasa por alto la posibilidad de que el usuario no tenga instalado las librerías de Tkinter y en consecuencia nuestra aplicación no podrá funcionar, lo mejor en este caso como en muchos otros es anticiparnos a los posibles errores que puedan ir surgiendo y manejar las excepciones de la siguiente forma:

Python 2.x

```
try:
    import Tkinter
except ImportError:
    raise ImportError("Se requiere el modulo Tkinter")
```

Python 3.x

```
try:
    import tkinter
except ImportError:
    raise ImportError("Se requiere el modulo tkinter")
```

Tambien podriamos ir un poco mas lejos y suponer que no sabemos que version esta usando el usuario y llevar el codigo de arriba para que sea compatible con ambas versiones de Python

```
import sys

PYTHON_VERSION = sys.version_info.major

if PYTHON_VERSION < 3:
    try:
        import Tkinter as tk
    except ImportError:
        raise ImportError("Se requiere el modulo Tkinter")
else:
    try:
        import tkinter as tk
    except ImportError:
        raise ImportError("Se requiere el modulo tkinter")
```

Otra forma igual de efectiva

```
try:
    import Tkinter as tk
except ImportError:
    import tkinter as tk
```

La forma mas elegante y eficiente para mi gusto (Aclaro que solo es mi humilde opinion), es a traves de una libreria externa llamada *six* que la pueden bajar e instalar desde aqui https://pypi.python.org/pypi/six. Con esta podemos obtener la primer forma mostrada pero mas compacta. Dependera de ustedes si van a agregar librerias externas a su aplicacion.

```
try:
    from six.moves import tkinter as tk
except ImportError:
    raise ImportError("Se requiere el modulo Tkinter")
```

De esta forma en caso de que el potencial usuario de nuestra preciada aplicación, no pueda ejecutarla pueda saber cual es motivo (en este caso es que no tenga instalado Tkinter) por el cual no pudo iniciarla.

Una vez importado el modulo Tkinter correctamente podemos utilizarlo para crear nuestra primera ventana de la siguiente forma:

```
from six.moves import tkinter as tk

root = tk.Tk()
root.mainloop()
```

Estas lineas son fundamentales, ya que de ellas dependerá gran parte del contenido así como pueden ser botones y menús, aunque mas adelante conforme vayamos viendo temas mas avanzados podremos hacer uso de otras herramientas y técnicas.

En la primera linea (No tengamos en cuenta la importación vista anteriormente) se crea un identificador que sera el que utilizaremos para referirnos a la ventana, en este caso lo llamamos 'root' y es una de las funciones mas importantes de Tkinter. Siempre que iniciamos un identificador que en este caso lo llamamos 'root' debemos cerrarlo para capturar como veremos mas adelante los eventos.

Nota: Para que el codigo escrito sea mantenible se utilizara de ahora en adelante el modulo *six*, esto permite ignorar la version de Python utilizada.

Nota: La variable 'root' usada para nombrar al identificador, puede ser reemplazada por cualquier otro nombre siempre y cuando se respeten las palabras reservadas de Python, aunque es muy utilizado usar el nombre 'root' para la ventana principal y puede ser una buena practica para que resulte mas familiar a los demás programadores que se quieran unir al desarrollo de nuestra aplicación.

Con esto ya tendremos una ventana vacía que nos servirá para comenzar a trabajar, a partir de ahora iremos ampliando el contenido mostrando los distintos widgets con los que contamos en Tkinter y luego para finalizar crearemos un ejemplo sencillo para unir lo se vio a lo largo de este material.

CAPÍTULO 11

Y que hay de las clases?

Los que estuvieron viendo otros tutoriales antes que este, habran notado el uso de clases para crear las interfaces. Esta convencion es la mas recomendada por muchos motivos

Veamos un ejemplo introductorio a esta idea y analicemosla:

Lista 11.1: Ejemplo minimo de tk usando *class*

```
1  from six.moves import tkinter as tk
2
3  class UI(tk.Frame):
4      """Docstring."""
5
6      def __init__(self, parent=None):
7          tk.Frame.__init__(self, parent)
8          self.parent = parent
9          self.init_ui()
10
11     def init_ui(self):
12         """Aqui colocariamos los widgets."""
13         self.parent.title("Un titulo para la ventana")
14
15 if __name__ == "__main__":
16     ROOT = tk.Tk()
17     ROOT.geometry("800x600")
18     APP = UI(parent=ROOT)
19     APP.mainloop()
20     ROOT.destroy()
```

En la linea 15 le estamos diciendo a Python que cuando el usuario ejecute nuestro archivo cree un objeto tkinter, en el vemos un codigo muy similar al conocido

CAPÍTULO 12

Las distintas librerias tk, ttk y tix

CAPÍTULO 13

Widgets

En esta sección no nos centraremos en el posicionamiento y diseño de los elementos que compondrán nuestra interfaz gráfica, lo dejaremos para la siguiente sección *Gestión del diseño* en donde se vera mas detenidamente este asunto.

Sobre las opciones que soportar los widgets seran analizados en una seccion aparte, debido a que el funcionamiento de muchas de ellas son similares en cada widget, en caso de presentar alguna diferencia la misma sera mencionada.

En la siguiente tabla veremos que widgets tenemos disponibles para las tres librerias que vimos anteriormente, los Que tienen enlace es porque ya se encuentran documentados en la guia.

tk	ttk	tix
Button	Button	Ballon
Canvas	Checkbutton	ButtonBox
Checkbutton	Combobox	CheckList
Entry	Entry	ComboBox
Frame	Frame	Control
Label	Label	DialogShell
LabelFrame	LabeledScale	DirList
Listbox	Labelframe	DirSelectBox
Menu	Menubutton	DirSelectDialog
Menubutton	Notebook	DirTree
Message	OptionMenu	ExFileSelectBox
OptionMenu	Panedwindow	ExFileSelectDialog
PanedWindow	Progressbar	FileEntry
Radiobutton	Radiobutton	FileSelectBox
Scale	Scale	FileSelectDialog
Scrollbar	Scrollbar	Hlist
Spinbox	Separator	InputOnly
Text	Sizegrip	LabelEntry
	Treeview	LabelFrame
		ListNoteBook
		Meter
		NoteBook
		OptionMenu
		PanedWindow
		PopupMenu
		ResizeHandle
		ScrolledHList
		ScrolledListBox

Continúa en la página siguiente

Tabla 13.1 – proviene de la página anterior

tk	ttk	tix
		ScrolledText
		ScrolledTList
		ScrolledWindow
		Select
		Shell
		StdButtonBox
		Tlist
		Tree

13.1 Etiquetas [Label]

Para comenzar a llenar esa vacía ventana que acabamos de crear vamos a nombrar a las etiquetas o mas conocidas como *'label'* por su nombre en ingles, las podemos encontrar desde interfaces gráficas hasta en formularios en HTML y son sumamente útiles en la construcción de interfaces y formularios, al igual que cuando creamos nuestra ventana vacía declarándola de la siguiente forma *"root = Tkinter.Tk()"* tendremos que hacerlo con nuestra etiqueta, para esto necesitamos escoger un nombre en mi caso escogeré **"etiqueta"** para declararla y deberemos llamar al widget "Label" en vez de la función "Tk" como habíamos hecho anteriormente, como para orientarnos con lo ya visto quedando por ahora algo así:

```
etiqueta = Tkinter.Label()
```

Una vez creada la estructura básica de la etiqueta, tendremos que llenar esa paréntesis. Primero tendremos que indicar a donde pertenece ese 'label' indicando el identificador de la ventana que en nuestro caso es 'root', separado por una coma escribiremos lo siguiente **text=** en donde seguido del signo igual incluiremos el texto que deseamos que incluya la etiqueta, la cual puede ser un texto (que tendremos que encerrar entre comillas al igual que cuando usamos la sentencia print) o una variable que contenga al texto. Para verlo un poco mas claro juntemos todo en dos ejemplos, uno donde muestre un texto fijo que le indiquemos a la etiqueta y otro usando una variable

- **Ejemplo numero 1**:

```
etiqueta = Tkinter.Label(root, text="Probando Label")
```

- **Ejemplo numero 2**:

```
import getpass
texto = "Bienvenido %s a nuestra guia:\t" % getpass.getuser()
etiqueta_2 = Tkinter.Label(root, text=texto)
```

Expliquemos los dos ejemplos, en el primero usamos un texto fijo no dinámico para utilizar como valor de la etiqueta en cambio en el segundo estamos usando una variable en donde su valor varia dependiendo de la salida que obtengamos al ejecutar *'getpass.getuser()'* que obtiene el nombre de usuario que tengamos en nuestra PC, como podemos observar su funcionamiento no varia mucho de lo que nos tiene acostumbrados Python, así que manejarnos con Tkinter es solamente aprendernos un par de nombres de sus widgets como para poder empezar a utilizarlo, el resto sigue siendo código Python.

Pero (siempre hay un pero) antes de poder ejecutar este ejemplo en nuestra consola tendremos que declarar este widget en nuestra ventana dándole una ubicación en la misma, sino hacemos esto nuestra etiqueta simplemente no aparecerá en nuestra ventana, va a existir pero no se visualizara. Para hacerlo tenemos tres formas (Esto se vera mas detenidamente mas adelante) una es usando **'pack'** y las otras dos son **'grid'** y **'place'**. Como esto se vera mas adelante haré una breve síntesis de las tres opciones que tenemos:

Por hacer

Escribir sintesis

- 'pack':
- 'grid':
- 'place:

Como en este ejemplo solo colocaremos una etiqueta no es necesario recurrir al uso de 'grid' para posicionarlo, así que usaremos 'pack' en su lugar. Pero lo mas importante es saber como se declara esta etiqueta en nuestra ventana y poder ver nuestro primer widget en funcionamiento, para lograr esto deberemos usar la siguiente forma:

```
etiqueta.pack()
```

Como vemos es muy simple solo debemos usar el nombre que utilizamos para declarar nuestra etiqueta seguida de la forma que vayamos a utilizar para posicionarla, pero para los que no desean esperar a leer la siguiente sección para ver como se utiliza 'grid' les dejo la misma etiqueta pero posicionándola con 'grid' en lugar de usar 'pack' como ya lo habíamos visto:

```
etiqueta.grid(row=1, column=1)
```

Como se puede observar esta otra forma debimos indicarle dos parámetros que sirven para especificar la posición del widget, en este caso nuestra etiqueta. El primer parámetro es *'row'* que se traduce como fila al español y el segundo es *'column'* o columna en español con ellos podemos indicar en que columna y fila deberá colocarse el widget, aunque por ahora no profundizaremos entrando en detalles en la forma que trabaja cada uno ni todos los parámetros que acepta cada una.

Si ahora escribiéramos todo lo que vimos en nuestra consola tendríamos que ver nuestra primera etiqueta en funcionamiento, contenido dentro de una ventana quedando todo el código como el siguiente:

```
from six.moves import tkinter as tk

class UI(tk.Frame):
    """Docstring."""

    def __init__(self, parent=None):
        tk.Frame.__init__(self, parent)
        self.parent = parent
        self.init_ui()

    def init_ui(self):
        """Aqui colocariamos los widgets."""
        self.parent.title("Un titulo para la ventana")

        etiqueta = tk.Label(self.parent, text="Ejemplo simple de Label")
        etiqueta.pack()

if __name__ == "__main__":
    ROOT = tk.Tk()
    ROOT.geometry("300x100")
    APP = UI(parent=ROOT)
    APP.mainloop()
    ROOT.destroy()
```

Y así es como quedara nuestro ejemplo terminado:

Habiendo ya escrito esas lineas se da por finalizada la muestra de este primer widget, pero antes veamos algunos métodos que tenemos disponibles.

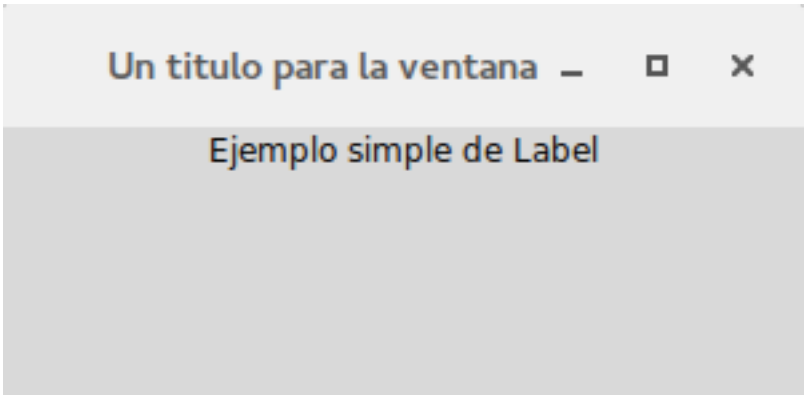


Figura 13.1: Ejemplo Basico de Label

13.1.1 Métodos

cget

cget(*option*)

Parámetros **option** (*str*) – El valor de la opcion a consultar.

Devuelve Devuelve la cadena con el valor de la opcion consultada

Tipo del valor devuelto str

Muestra

- **TclError** – si no existe la opcion
- **TypeError** – si no se pasa una opcion

Este método nos permite obtener el valor de determinada opción pasada al widget, supongamos que deseamos obtener el color de fondo (background) o el tipo de borde (relief) que tiene, para esto contamos con 'cget' el cual pasandole como una cadena el nombre de la opción nos devuelve su valor.

Por ejemplo si sobre nuestro widget llamáramos al método 'cget' con los siguientes argumentos veríamos algo así:

```
>>> etiqueta = tk.Label(self.parent, text="Ejemplo simple de Label")
>>> etiqueta.cget('background')
'#d9d9d9'
>>> etiqueta.cget('relief')
'flat'
>>> etiqueta.cget('text')
'Ejemplo simple de Label'
```

Así funcionaria de la misma forma para todas las opciones soportadas por el widget.

configure

Por hacer

Mejorar uso de Sphinx (http://sphinx-doc.org/domains.html#the-python-domain)

configure()

Devuelve Listado completo de las opciones y sus configuraciones disponibles

Tipo del valor devuelto dict

configure(*option*)

Parámetros **option** (*str*) – Opcion a consultar

Devuelve Devuelve la configuracion para la opcion pasada

Tipo del valor devuelto tuple

configure(*{option:value}*)

Parámetros **option** (*dict*) – Diccionario con las opciones a modificar

Tipo del valor devuelto None

El metodo *configure* es muy potente y hace de extension de *cget*, con el podemos obtener un valor si le pasamos como cadena la opcion a consultar al igual como lo hace *cget*, pero lo hace devolviendo una tupla con mas informacion, mas adelante en una seccion avanzada de la guia lo veremos mas detalladamente.

Si no pasamos un argumento nos mostrara los valores de todas las opciones existentes.

Por ultimo podemos pasarle un diccionario con una o mas opciones como llaves con los valores a establecer, a continuacion se muestra un ejemplo para cambiar el color de fondo de la etiqueta.

```
>>> etiqueta.cget('background')
'#d9d9d9'
>>> etiqueta.configure('background')
('background', 'background', 'Background', <border object at 0x559d5b771410>, '#d9d9d9')
>>> etiqueta.configure({'background': '#ccc'})
>>> etiqueta.cget('background')
'#ccc'
>>> etiqueta.configure()
{'highlightthickness': ('highlightthickness', 'highlightThickness', 'HighlightThickness', <pixel obje
'text': ('text', 'text', 'Text', '', 'Ejemplo simple de Label'),
'image': ('image', 'image', 'Image', '', ''),
'compound': ('compound', 'compound', 'Compound', <index object at 0x559d5b771980>, 'none'),
...
```

Por hacer

Mencionar metodos para ttk

13.2 Botones [Button]

Los botones pueden contener texto o imágenes y se les puede asociar funciones o métodos, que al hacer clic sobre ellos Tkinter se encargara de llamar automáticamente a ese método o función y ejecutara el código Python contenido dentro de ellos.

Un detalle es que solo puede utilizarse un tipo de fuente, pero ese texto puede ocupar varias lineas en caso de ser necesario. Además se puede subrayar el texto, un carácter o letra en particular, algo que es muy común cuando se desea decirle al usuario sobre la existencia de un atajo con el teclado.

Antes de crear a ese botón debemos declarar la función a la cual se llamara, porque como se imaginaran no podemos decirle al botón que ejecute una función que no existe, por eso vamos a escribir un par de lineas y crearemos una función simple como para verificar que dicho botón funciona correctamente.

```
def función():
  print "Excelente"
```

Con esas dos lineas nos va a servir para demostrar su funcionamiento, ahora si vamos por el botón.

A continuación les muestro el código con el cual vamos a trabajar:

```
from six.moves import tkinter as tk

def funcion():
    print("Excelente")

root = tk.Tk()
boton = tk.Button(root, text="Que te parece la guía?", command=funcion)
boton.pack()
root.mainloop()
```

En el se observa la ya conocida importación del modulo Tkinter, la función que ya habíamos declarado y la creación del botón. Como se puede ver la creación de un botón básico es muy similar al de una etiqueta pero se incluye un parámetro mas que nos dará la posibilidad de especificar dentro de el la función que utilizaremos; el parámetro mencionado es **'command'** y en el podemos llamar a funciones o métodos, en este caso llamamos a la función 'función' que va a imprimir en la consola la palabra *'Excelente'*.

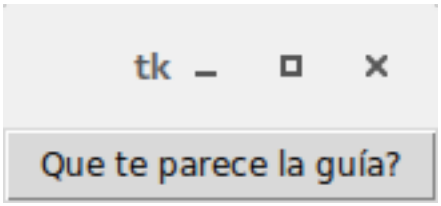


Figura 13.2: Boton basico

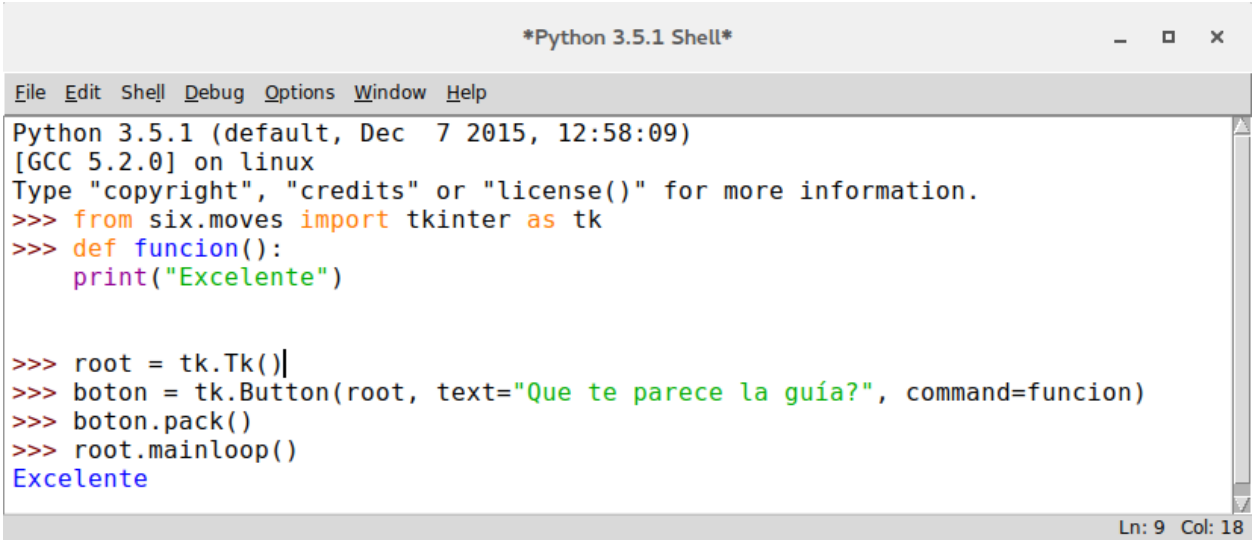


Figura 13.3: Comando basico en un boton

13.2.1 Métodos

cget

Este método cumple la misma función que se vio en el widget *Label*. *Ver aqui*.

configure

Este método cumple la misma función que se vio en el widget *Label*. *Ver aqui*.

invoke

invoke()

Devuelve Devuelve valor de retorno de la funcion asociada a metodo *command*

Tipo del valor devuelto srt or None

Este método invoca al comando asociado al botón, El valor que retorna es el valor de retorno del comando o una cadena vacía en caso de no existir un comando asociado. En caso de que el botón se encuentre desactivado este método se ignora.

```
>>> from six.moves import tkinter as tk
>>> def funcion():
...     print("Excelente guia")
...
>>> root = tk.Tk()
>>> boton = tk.Button(root, text="Probando el boton", command=funcion)
>>> boton.pack()
>>> boton.invoke()
Excelente guia
'None'
>>>
```

```
>>> from six.moves import tkinter as tk
>>> def funcion():
...     return "Excelente guia"
...
>>> root = tk.Tk()
>>> boton = tk.Button(root, text="Probando el boton", command=funcion)
>>> boton.pack()
>>> boton.invoke()
Excelente guia
>>>
```

```
>>> from six.moves import tkinter as tk
>>> root = tk.Tk()
>>> boton = tk.Button(root, text="Probando el boton")
>>> boton.pack()
>>> boton.invoke()
''
>>>
```

13.3 Cuadro de texto [Entry]

Por hacer

Enlazar con seccion options

Este widget nos permite tanto mostrarle información al usuario como también obtenerla de su parte, dotando a nuestra ventana de un widget muy poderoso y útil, siendo el primero de los widgets mencionados que nos posibilitara la interacción del usuario solicitando la entrada de datos. Como venimos haciendo en esta guía se mostraran algunos conceptos básicos que te permitirán empezar a utilizarlo, pero luego en la sección 'Opciones' se detallan las opciones que tienen disponibles al crear un cuadro de texto.

Este elemento es muy popular y sumamente útil así que deberemos de conocerlo y manejarlo, para esto comencemos explicando algunos conceptos básicos para entender mejor de se trata esto.

Para empezar a utilizar nuestro elemento 'Entry' podemos primero declarar una variable del tipo 'StringVar' o del tipo 'IntVar' (tenemos mas opciones disponibles pero se verán mas detenidamente en otra sección), estas variables capturan el texto que el usuario ingrese en el cuadro para luego poder trabajar con esos datos, aunque esta practica es muy recomendada y hasta imprescindible en ocasiones, no es un requisito obligatorio para utilizar nuestro cuadro de texto como es visto en algunos sitios web.

Por esto veremos primero un uso mas básico para luego entrar mas en detalle e incorporar estas variables a nuestro repertorio.

La forma en que se declara un cuadro de texto es muy simple:

```
campo_de_texto = tk.Entry(root)
```

Esa es su forma mas básica en donde como se observa no se llama a ninguna de las opciones que tenemos disponibles, antes de ver los métodos que podemos llamar para hacer uso de este widget veamos un ejemplo de su uso

```
>>> from six.moves import tkinter as tk
>>> root = tk.Tk()
>>> campo_de_texto = tk.Entry(root)
>>> campo_de_texto.pack()
>>> campo_de_texto.get()
''
>>>
```

Si escribiéramos un texto en el cuadro y escribimos en la consola lo siguiente

```
campo_de_texto.get()
```

estaríamos usando el método 'get' del widget Entry que nos permite obtener el texto que contenga el cuadro de texto, como podemos observar en el siguiente ejemplo

```
>>> from six.moves import tkinter as tk
>>> root = tk.Tk()
>>> campo_de_texto = tk.Entry(root)
>>> campo_de_texto.pack()
>>> # En este caso escribi "Hola Mundo!!!"
>>> campo_de_texto.get()
'Hola Mundo!!!'
>>>
```

Seguramente ese rectángulo en nuestra ventana de poco serviría si no pudiésemos interactuar con este. A continuación veremos los métodos a lo podemos llamar para trabajar con este widget.

Una observación que se debe hacer, es que para escribir varias lineas de texto se debe utilizar el widget 'Text' que esta pensado para esto, que sera visto mas adelante.

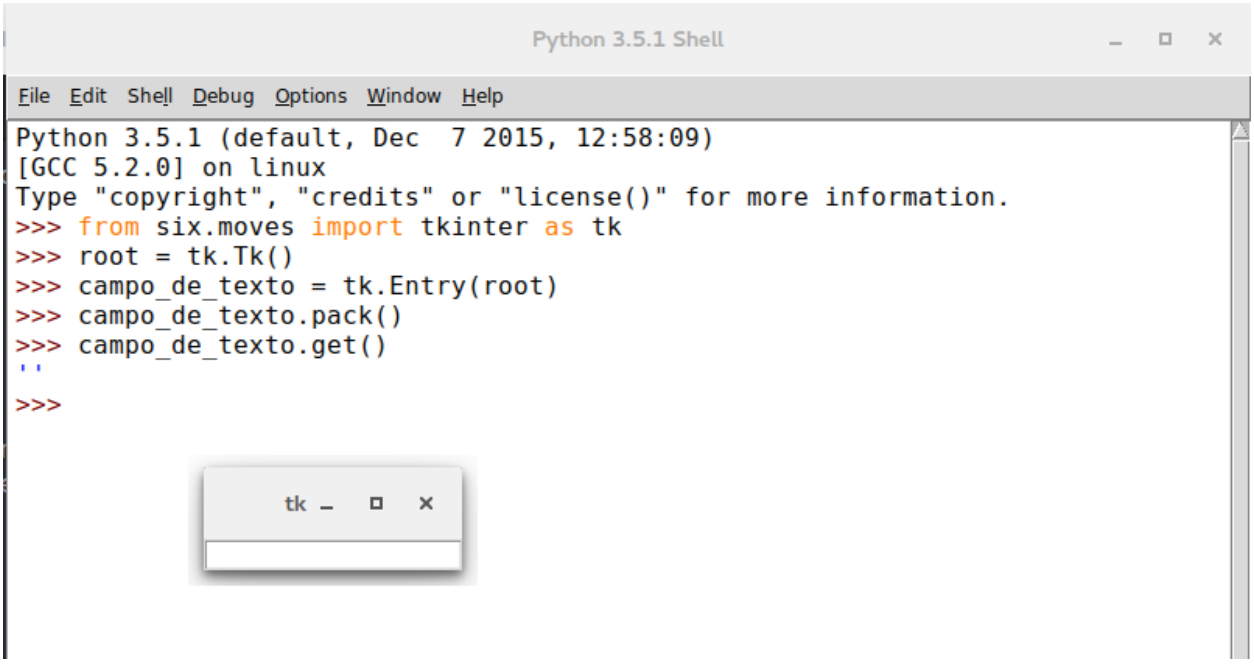


Figura 13.4: Ejemplo basico de Entry devolviendo una cadena vacia

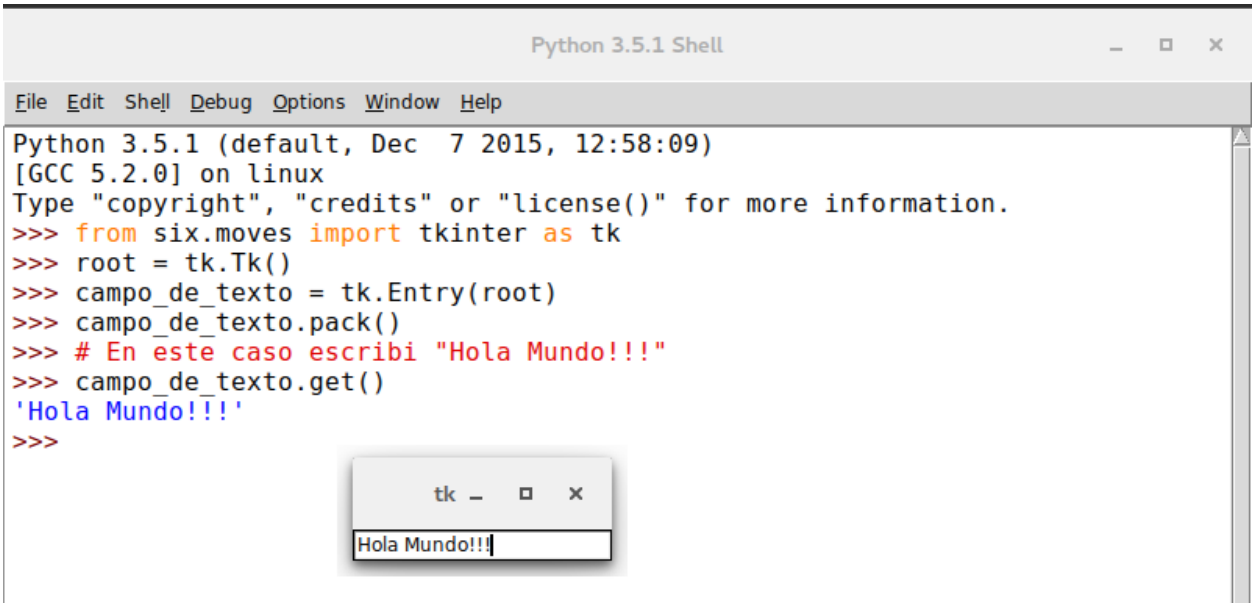


Figura 13.5: Ejemplo de como obtener texto con el metodo **get**

13.3.1 Índices

Un concepto que hay que tener en cuenta antes de empezar a utilizar algunos métodos disponibles para 'Entry' son los índices, estos nos facilitan el insertado y eliminado de caracteres entre otras cosas, estableciendo desde y hasta donde deseamos por ejemplo borrar o insertar texto en el cuadro de texto. Para conseguir esto contamos con varias formas que las iremos viendo de a una para luego verlas dentro de los métodos que las soportan y ya con unos ejemplos para afianzar los conocimientos.

Índices numéricos

Estos funcionan como los índices de las listas de Python así que su aplicación no debería ser complicada, el primer carácter comenzara a contar desde 0 en adelante. Para mas referencia repase los conceptos básicos de listas y tuplas en Python.

ANCHOR

Por hacer

Enlazar con metodos select_from y select_adjust

Este índice corresponde al inicio del texto que se encuentre seleccionado, en caso de existir un texto seleccionado. Se puede usar select_from y select_adjust para alterar el comportamiento del mismo.

END

También podemos marcar desde un determinado carácter hasta el ultimo sin conocer la cantidad de caracteres que componen a ese campo de texto, esto es gracias a 'END' que haría de comodín para manejarnos de una forma mas sencilla. El uso de los índices (0, END) equivaldría a seleccionar todo el texto disponible.

INSERT

Este índice corresponde a la posición actual de donde se encuentra el cursor. Este índice se puede trabajar junto con el método 'icursor' para alterar el lugar del cursor.

Obviamente cuando nos referimos a cursor nos referimos a la barra vertical que nos aparece cuando escribimos no al cursor del mouse (Ver mas información aquí Cursor de texto).

sel.first y sel.last

Así como vimos opciones para trabajar con índices sobre el texto del cuadro, tenemos dos índices mas específicos que nos permiten trabajar únicamente con el texto que se encuentre seleccionado. Con sel.first indicamos que deseamos trabajar con el primer carácter del texto que se encuentre seleccionado y con sel.last indicamos que deseamos trabajar con el ultimo carácter de la selección.

@number

Otras de la posibilidades disponibles es usar la posición del cursor del mouse para indicar el índice.

La forma en que trabaja es a través de la siguiente sintaxis

@x

Donde 'x' es un valor en píxeles con respecto al borde izquierdo del campo de texto. Por ejemplo '@0' indica el carácter mas a la izquierda del campo del texto.

Nota: Una observación que se puede hacer sobre los índices es que pueden usarse abreviaturas de los nombres, por ejemplo en vez de usar 'end' se puede utilizar 'e'. Como opinión personal no recomiendo el uso de esta practica ya que puede quitar legibilidad al código y generar malas costumbres.

Estos índices nos permiten tener un mayor control sobre sobre los cuadros de texto y lo veremos mas claro en la sección siguiente cuando veamos los métodos disponibles para este.

13.3.2 Métodos

Como habíamos comentando cuando estábamos viendo el widget Entry, dijimos que existían métodos que nos permitían trabajar con este, los cuales iremos viendo a continuación.

bbox

Por hacer

Escribir descripcion

---

cget

Este método cumple la misma función que se vio en el widget *Label*. *Ver aqui*.

configure

Este método cumple la misma función que se vio en el widget *Label*. *Ver aqui*.

delete

delete(*first*, *last=None*)

Parámetros

- **first** (*indice*) – Indica en inicio del indice
- **last** (*indice*) – Indica el fin del indice

Tipo del valor devuelto None

Elimina uno o mas elementos del campo de texto, este método recibe dos argumentos uno es el inicio desde donde se desea eliminar el contenido y el segundo argumento es hasta donde deseamos eliminar, al igual que en Python debemos comenzar el índice contando desde 0. Por ejemplo para eliminar de la siguiente cadena

'0123456789'

de los caracteres del 0 al 5, los índices que le deberíamos de dar serian (0, 6). Esto no es nada nuevo a lo ya venimos viendo en este lenguaje de programación.

También podemos pasarle un solo argumento para borrar un solo carácter, veamos estos dos a través de unos ejemplos.

Con un solo argumento:

```
>>> from six.moves import tkinter as tk
>>> root = tk.Tk()
>>> campo_de_texto = tk.Entry(root)
>>> campo_de_texto.pack()
>>> campo_de_texto.get()
'0123456789'
>>> campo_de_texto.delete(0)
>>> campo_de_texto.get()
'123456789'
>>>
```

Con dos argumentos:

```
>>> from six.moves import tkinter as tk
>>> root = tk.Tk()
>>> campo_de_texto = tk.Entry(root)
>>> campo_de_texto.pack()
>>> campo_de_texto.get()
'0123456789'
>>> campo_de_texto.delete(0, 6)
>>> campo_de_texto.get()
'6789'
>>>
```

Otra opción que tenemos es borrar desde un inicio fijado por nosotros hasta el final del texto disponible en el campo de texto gracias a un índice que nos da Tkinter que es 'END', con este podemos de la siguiente forma borrar desde el inicio hasta el final sin necesitar conocer cuantos caracteres componen el texto contenido en el widget

```
campo_de_texto.delete(0, tk.END)
```

Aunque no es necesario comenzar desde el inicio también podemos borrar desde el quinto carácter hasta el ultimo

```
campo_de_texto.delete(5, tk.END)
```

También podemos utilizar los índices ya vistos en la sección 'Índices'.

get

get()

Devuelve Devuelve el texto del campo de texto

Tipo del valor devuelto str

Obtiene el contenido del campo de texto. Algo muy necesario para nosotros es poder obtener el texto contenido dentro del campo de texto, para conseguir esto podemos hacernos del método 'get' que nos lo permitirá, este nos retornara el texto como una cadena. Veamos con un ejemplo escribiendo cualquier texto y llamamos al método 'get'

```
>>> from six.moves import tkinter as tk
>>> root = tk.Tk()
>>> campo_de_texto = tk.Entry(root)
>>> campo_de_texto.pack()
>>> # Ahora deben escribir el texto
>>> # en el campo de texto
>>> campo_de_texto.get()
'Mostrando el uso de get'
>>>
```

El valor que nos retorne siempre sera como una cadena y es muy simple comprobarlo, coloquemos un texto en nuestro campo de texto, en mi caso escribiré la cadena 'Viva Python!!!' y almacenemos su valor en una variable para luego analizarla.

```
>>> var = campo_de_texto.get()
>>> var
'Viva Python!!!'
>>> isinstance(var, str)
True
>>> var = campo_de_texto.get()
>>> var
'1'
>>> isinstance(var, str)
True
>>>
```

Mas adelante veremos como validar nuestros campos de textos.

icursor

icursor(*index*)

Parámetros **index** (*indice*) – Indica donde colocar el cursor

Tipo del valor devuelto None

Nos permite mover el cursor hasta la posición indicada, cuando vimos acerca de los índices vimos a INSERT y mencionamos a 'icursor'. Este método nos permite cambiar a donde deseamos que cambie la posición del cursor y junto a INSERT podemos llevar nuestra aplicación a otro nivel de interacción con el usuario.

Un ejemplo simple seria el siguiente

```
>>> entrada.icursor(0)
```

```
>>> entrada.icursor(tk.END)
```

Recuerden de haber escrito un texto en el campo y que tenemos a nuestra disposición los índices para indicar la posición del cursor.

index

index(*index*)

Indica el valor numerico del indice indicado. La utilidad de 'index' esta en que nos permite saber por ejemplo cual es el indice del ultimo caracter, ya vimos que con END podiamos ir al ultimo caracter del cuadro de texto, pero como saber cuantos caracteres tiene esa cadena sin recurir a 'len()', ahi entra 'index' que nos permite obtener esos valores. Aclaremos estos conceptos con unos sencillos ejemplos.

```
>>> campo_de_texto.get()
'ABCD'
>>> campo_de_texto.index(tk.END)
4
>>>
```

Ahora probemos poniendo el cursor despues del tercer caracter y veamos si podemos obtener su posicion

```
>>> campo_de_texto.index(tk.INSERT)
3
>>>
```

Asi como funciono con 'end' y 'insert' tambien deberia hacerlo con 'anchor', asi que volvamos a la consola. Para el primer ejemplo seleccionemos desde el indice 4 al 0, es decir seleccionemos todo el texto de derecha hacia izquierda

```
>>> campo_de_texto.index(tk.ANCHOR)
4
>>>
```

Y ahora hagamos algo similar pero seleccionando de izquierda hacia derecha

```
>>> campo_de_texto.index(tk.ANCHOR)
0
>>>
```

Como vemos pasandole el indice 'anchor' podemos obtener el indice numerico desde donde se inicio la seleccion del texto.

Y como ultimo ejemplo como resultaria logico si le pasamos como argumento un indice numerico nos deberia devolver ese mismo numero

```
>>> campo_de_texto.index(3)
3
>>>
```

Salvo que le indiquemos un numero mayor a la cantidad de caracteres disponibles, que en ese caso nos devolvera el valor del ultimo indice

```
>>> campo_de_texto.index(9)
4
>>>
```

insert

insert(*index*, *string*)

Parámetros

- **index** (*indice*) – Indica donde colocar la cadena o caracter
- **string** (*str*) – Cadena o caracter a ingresar

Tipo del valor devuelto None

Permite la inserción de texto. Supongamos que necesitamos insertar una cadena, un numero o un carácter en un campo de texto, para conseguirlo es tan simple como llamar a este método pasándole dos argumentos como parámetros. De los dos argumentos mencionados uno es el índice de donde deseamos insertar el texto y el segundo es la cadena o caracter.

En este ejemplo insertaremos la cadena "GNU/" desde el índice 0

```
>>> from six.moves import tkinter as tk
>>> root = tk.Tk()
>>> campo_de_texto = tk.Entry(root)
>>> campo_de_texto.pack()
>>> campo_de_texto.get()
'Linux'
>>> campo_de_texto.insert(0, "GNU/")
>>> campo_de_texto.get()
'GNU/Linux'
>>>
```

xview

xview()

Devuelve

Tipo del valor devuelto tuple

Este comando se utiliza para consultar y cambiar la posición horizontal del texto en la ventana del widget. Puede tomar cualquiera de las siguientes formas:

Por hacer

Escribir metodo

13.3.3 Validando un campo de texto

Con lo que vimos ahora estamos mas cerca de dominar los cuadros de texto, aunque no hay que apresurarnos que nos queda mucho camino por delante.

Para referencias acerca de textvariable vea la sección "Variables".

Por hacer

Primero escribir seccion *Variables*

13.4 Variables

Acá es donde se muestra una de las opciones que disponemos en varios widgets y es 'textvariable' que tomara como valor la variable del tipo StringVar que habíamos mencionado en 'Entry', esta nos permitirá trabajar con varios métodos en Entry que se los iré mostrando y los iremos viendo con ejemplos. Cabe aclararse que el uso de estas variables no se limitan solamente a la opcion 'textvariable' como lo podremos iremos viendo en otros ejemplos mas adelante.

Tengan en cuenta que StringVar no forma parte de Python sino de Tkinter así que la importación de este es fundamental para poder utilizarlo, aunque para algunos resulte una obviedad nunca esta de mas remarcarlo ya que si utilizamos la forma *"from Tkinter import *"* estos detalles no se observan y pueden generar confusión cuando se ven los ejemplos.

Por hacer

Enlazar BooleanVar, DoubleVar, IntVar y StringVar con ancla

Como habíamos mencionado contamos con cuatro clases de variables con Tkinter las cuales son BooleanVar, DoubleVar, IntVar y StringVar, esta ultima suele ser la mas vista y utilizada en los tutoriales disponibles en internet pero no por esto mas importante que las demás.

Para comenzar a utilizarlo veamoslo con un ejemplo simple, supongamos que tenemos un cuadro de texto y queremos almacenar el texto ingresado en una variable, el problema con usar el metodo 'get' del widget 'Entry' es que, lo que se almacenaria seria el valor que contiene el cuadro al momento de llamar al metodo lo cual es util, pero hay veces que queremos las modificaciones que se puedan ir realizando a dicho texto. En estos casos es donde se nos brindan herramientas como estas variables donde las modificaciones que se hacen en el cuadro de texto se ven reflejadas en la variable, mas adelante se vera que nos estan limitadas a cuadros de texto sino que contaremos con otros widgets que pueden hacer uso de estas.

Un ejemplo sensillo para verlo en accion:

```
>>> # Primero creamos nuestra ventana vacia
>>> from six.moves import tkinter as tk
>>> root = tk.Tk()
>>> # Definimos nuestra variable del tipo "StringVar"
>>> variable = tk.StringVar()
>>> variable
<tkinter.StringVar object at 0x7fe0de0dd630>
>>> # Creamos nuestro campo de texto pasandole nuestra variable
>>> campo_de_texto = tk.Entry(root, textvariable=variable)
>>> campo_de_texto.pack()
>>> # Insertamos un poco de texto
>>> campo_de_texto.insert(0, "Probando")
>>> # Aca llamamos al metodo 'get' de 'Entry'
>>> campo_de_texto.get()
'Probando'
>>># y aca al metodo 'get' pero de nuestra variable para obtener su valor
>>> variable.get()
'Probando'
>>> # Y para repetir su funcionamiento agregamos unpoco mas de texto
>>> campo_de_texto.insert('end', " como funciona")
>>> campo_de_texto.get()
'Probando como funciona'
>>> variable.get()
'Probando como funciona'
>>>
```

Nota: Si desean ver como funcionan mas en detalle estas variables pueden ver estos objetos desde el codigo fuente de la libreria Tkinter, para esto deben dirigirse a la carpeta donde se encuentra instalado Python y ahi van a la carpeta 'Lib' y luego a la carpeta 'lib-tk' en el caso Python 2.x y para Python 3.x deberan buscar en la carpeta 'tkinter', alli se encuentra el codigo fuente de Tkinter, la clase que contiene al objeto 'StringVar' se encuentra dentro del archivo 'Tkinter.py' en Python 2.x y '__init__.py' para Python 3.x que es el que se carga cuando hacemos 'import Tkinter'

Pudimos notar en ese ejemplo el uso de un metodo llamado 'get' el cual esta disponible para las cuatro tipos de variables, al igual que otros metodos que iremos viendo de a uno.

13.4.1 StringVar

Por hacer

- Especificar excepciones posibles
- Terminar espeficacion para master

StringVar(*master=None*, *value=None*, *name=None*)

Parámetros

- **master** –
- **value** (*str or None*) – Un valor predefinido para la variable
- **name** (*str or None*) – Un nombre para la variable

Empecemos viendo esa linea donde creamos esa variable llamada 'variable', donde a 'StringVar' no le pasamos ningun argumento, si se dieron una vuelta por el codigo fuente de tkinter habran visto cuando se inicializa el objeto se ve que toma 3 parametros con valores 'None', analicemos su funcionamiento

master

Por hacer

Generar documentacion

value

Nos permite asignar un valor a la variable cuando esta es creada

```
>>> variable = Tkinter.StringVar(value="Hola")
>>> variable.get()
'Hola'
```

name

Por hacer

Ampliar informacion

Nos permite darle un nombre opcional, por defecto Tkinter le un nombre del tipo PY_VARnum, donde 'num' es reemplazado por un numero que varia dependiendo de la cantidad de variables que hayan sido creadas. Este valor es usado internamente por el lenguaje Tcl por lo que no nos interesara modificar su valor. Para conocer su valor simplemente llamamos al atributo '_name' que nos mostrara el valor de este.

```
>>> variable = tk.StringVar()
>>> variable._name
'PY_VAR0'
>>> variable = tk.StringVar(name="var", value="Hola")
>>> variable._name
'var'
```

Ahora que ya sabemos como y con que crear nuestra variable veamos los metodos que tememos disponibles.

get

get()

Devuelve El contenido de la variable

Tipo del valor devuelto str

Aunque ya vimos a 'get' no esta demas mencionarlo nuevamente ya que este seguramente sera el mas utilizado junto a 'set', *'get'* nos permite obtener el valor contenido en la variable.

```
>>> variable.get()
'Hola'
```

set

set(*value*)

Parámetros **value** (*str*) – El texto a ingresar o cambiar para la variable

Como se menciono en la descripcion de 'get' estos forman una gran pareja resultando en dos metodos basicos muy utiles, este metodo en particular nos permite modificar el contenido

```
>>> variable.set("Valor que deseamos guardar")
>>> variable.get()
'Valor que deseamos guardar'
```

Aunque lo vayamos a usar para almacenar cadenas de texto, tambien podemos pasarle un entero y nos lo guardaria como una cadena, si le pasamos un boleano lo almacenaria como una cadena con un 1 o un 0.

trace_variable

trace_variable(*mode*, *callback*)

Parámetros

- **mode** (*str*) – El modo de captura
- **callback** –

Devuelve Devuelve el indentificador interno de tk

Tipo del valor devuelto str

Ya vimos dos metodos sencillos, pero ahora mostraremos tres con un uso un poco mas avanzado pero una vez que entendamos como funcionan nos permitiran conseguir resultados bastantes avanzados. En primer lugar tenemos a *'trace_variable'* el cual nos permite definir una funcion a la cual llamar cuando se realice una determinada accion sobre la variable para ello cuenta con dos parametros uno de ellos es 'mode' que nos pide en que modo de captura es que llamara a la funcion, puede tomar tres valores 'w', 'r' y 'u' donde 'w' es para cuando se escribe en la variable, 'r' corresponde a la lectura de la misma y por ultimo 'u' que es llamado cuando la variable es eliminada.

Por hacer

Crear ejemplos

---

trace_vdelete

Mientras que con 'trace_variable' podiamos definir una funcion para determinados eventos tambien necesitabamos una herramienta que nos permitiera quitar esa respuesta para ese evento y es cuando entra 'trace_vdelete'. Su funcionamiento es es muy simple este toma un modo que ya habiamos visto y ademas nos pide el valor que nos devolvio 'trace_variable' al momento de ser ejecutado, veamoslo con un pequeño ejemplo y luego retomamos su explicacion

---

Por hacer

Crear ejemplo

---

trace_vinfo

Por hacer

Generar documentacion

13.4.2 Mas variables

Para finalizar veremos la diferencia entre las cuatro variables disponibles, ya vimos como crear y trabajar con las variables del tipo 'StringVar' pero que paso con el resto. Trabajar con 'StringVar' o con las demas es exactamente igual, las distinguimos por el tipo de valores que manejan, por ejemplo si necesitamos trabajar con cadenas de texto debemos utilizar a 'StringVar' como lo veniamos haciendo, si se trata de numeros enteros utilizaremos a 'IntVar', tambien puede ser el caso de que queremos trabajar con numeros pero con decimales para esto tenemos a 'DoubleVar' y por ultimo esta 'BooleanVar' que maneja los valores booleanos como *True* o *False*

La siguiente tabla es a modo de resumen:

Nombre	Tipo de valor en Python
StringVar	str
IntVar	int
DoubleVar	float
BooleanVar	bool

Y eso seria todo por el momento, el resto depende de ustedes y conforme lo vayan necesitando lo iran dominando.

13.5 Casillas de verificacion [Checkbutton]

Los que tengan conocimiento en interfaces o diseño web seguramente conocerán a los famosos 'checkbox' que en Tkinter tienen el nombre de 'checkbutton', para los nos los conocen se tratan de casillas de verificación cuya función es la de permitir seleccionar o deseleccionar, activando o desactivando funciones u opciones. A quien no le a tocado activar ese pequeño cuadrado al lado de un texto que nos dice "He leído y acepto los términos y condiciones del sitio.", aunque haya aplicaciones que puedan no requerirlo no significa que se pueda prescindir de ellos ya que su uso esta en gran parte de las aplicaciones de uso cotidiano desde nuestros editores o entornos de desarrollo cuando programamos hasta en las utilidades de configuración de nuestros sistemas operativos, aunque repetimos su uso nos es obligatorio siempre es bueno conocer las herramientas que tenemos disponibles.

Como venimos haciendo escribamos un poco de codigo y luego vayamos viendo que se escribio

```
>>> from six.moves import tkinter as tk
>>> root = tk.Tk()
>>> int_var = tk.IntVar()
>>> int_var.get()
0
>>> check = tk.Checkbutton(root, text="Probando", variable=int_var)
>>> check.pack()
>>> # Hacemos click en el checkbutton activandola
>>> int_var.get()
1
>>> # Ahora desmarcamos la opcion del checkbutton
>>> int_var.get()
0
>>>
```

```
>>> from six.moves import tkinter as tk
>>> root = tk.Tk()
>>> bool_var = tk.BooleanVar()
>>> bool_var.get()
0
>>> check = tk.Checkbutton(root, text="Probando", variable=bool_var)
>>> check.pack()
>>> bool_var.get()
True
```

```
>>> bool_var.get()
False
>>>
```

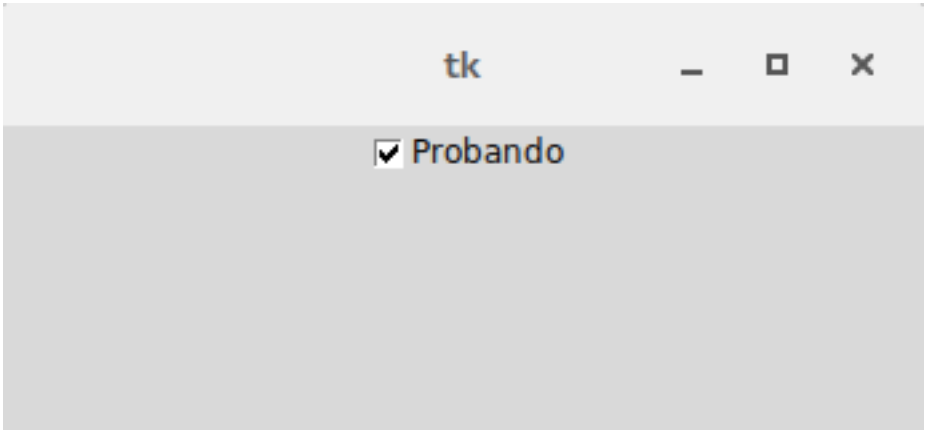


Figura 13.6: Ejemplo basico de Checkbutton

Por hacer

Enlazar 'IntVar' y 'BooleanVar' con sus secciones

En los dos ejemplos establecemos una variable que puede ser del tipo 'IntVar' o 'BooleanVar', las dos cumplen con su trabajo aunque trabajen con distintos tipos de datos consiguen el mismo resultado, establecida nuestra variable pasamos a crear y mostrar nuestra casilla de verificación y con ella marcándola y desmarcandola vemos como el valor de la variable alterna de True a False en el caso de 'BooleanVar' o de 0 a 1 en el caso de'IntVar'. Con este ejemplo podemos tener una idea de como trabaja 'Checkbutton' pero aun nos quedan ver algunos métodos para volverlo mas interesante.

13.5.1 Métodos

cget

Este método cumple la misma función que se vio en el widget *Label*. *Ver aqui*.

configure

Este método cumple la misma función que se vio en el widget *Label*. *Ver aqui*.

deselect

deselect()

Tipo del valor devuelto None

Desmarca la selección de la casilla de verificación y establece el valor de la variable asociada a False. Usarlo es muy simple, para verlo deberíamos crear un 'Checkbutton' básico marcarlo y luego llamar al método 'deselect'

```
>>> checkbutton.deselect()
```

flash

invoke

Por hacer

Completar valor de retorno

---

invoke()

Devuelve

Tipo del valor devuelto str

Este comando tiene la particularidad de realizar la misma acción que el método 'toggle' pero se diferencia porque 'invoke' se comporta como si el usuario hubiera hecho clic sobre el 'Checkbutton', con esto nos referimos a que si nuestra casilla de verificación tuviera asociado un comando a través de la opción 'command' esta se ejecutaría a diferencia de los métodos 'select', 'deselect' y 'toggle' que lo ignoran.

```
>>> checkbutton.invoke()
```

select

select()

Asi como contamos con una forma de desmarcar nuestro 'Checkbutton' con 'select' podemos marcarla.

```
>>> checkbutton.select()
```

toggle

toggle()

Este ultimo metodo une a 'select' y 'deselect', nos permite alternar su estado actualizando el valor de su variable asociada, por ejemplo si la casilla se encontrara marcada al llamar a 'toggle' esta se desmarcaria y si volvieramos a llamar a 'toggle' esta se volveria a marcar.

```
>>> checkbutton.toggle()
```

13.6 Botones de opción [Radiobutton]

Dentro de la familia de los botones acabamos de ver a *Checkbutton* y ahora le toca a *Radiobutton*. Estos dos se diferencian no solo en su forma sino que en *Radiobutton* solo uno de ellos puede estar seleccionado en la practica seria una opción cuando tenemos un tipo de selección de uno de muchos. Cada *Radiobutton* puede contener texto e imágenes y ser asociado a una función o método, que cuando se selección Tkinter llamaría automáticamente a esa función al igual que como lo haría con un botón.

Para conseguir este comportamiento hay que asegurarse de que todos los *Radiobutton* apunten a la misma variable.

Veamos un ejemplo sencillo para entenderlos un poco mas. Tenemos que tener en cuenta que opciones tendríamos que pasarles para poder utilizarlos, por un lado un texto o imagen para diferenciarlos y por el otro lado el valor que tendrían y una variable a la que estuvieran asociados. Como habíamos mencionado la variable debe coincidir en todos los *Radiobutton* del grupo y un valor único para cada uno. Ahora si pasemos al ejemplo:

```
>>> from six.moves import tkinter as tk
>>> root = tk.Tk()
>>> variable = tk.StringVar()
>>> def prueba():
...     print("Se ha elegido la opcion" + variable.get())
...
>>> radiobutton1 = tk.Radiobutton(text="Opcion 1", variable=variable, value=1, command=prueba)
>>> radiobutton2 = tk.Radiobutton(text="Opcion 2", variable=variable, value=2, command=prueba)
>>> radiobutton3 = tk.Radiobutton(text="Opcion 3", variable=variable, value=3, command=prueba)
>>> radiobutton1.pack()
>>> radiobutton2.pack()
>>> radiobutton3.pack()
>>> variable.get()
''
>>> # Ahora iremos seleccionando las distintas opciones
...
>>> Se ha elegido la opcion 1
Se ha elegido la opcion 2
Se ha elegido la opcion 3
>>>
```

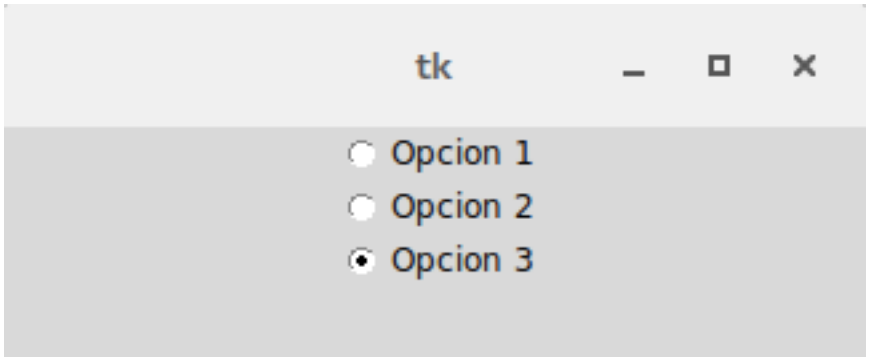


Figura 13.7: Ejemplo basico de Radiobutton

Como se habrán dado cuenta solo permite elegir una de las tres opciones, esto es debido a que como se había mencionado apuntan a la misma variable.

13.6.1 Métodos

cget

Este método cumple la misma función que se vio en el widget *Label*. *Ver aqui*.

configure

Este método cumple la misma función que se vio en el widget *Label*. *Ver aqui*.

deselect

Este método cumple la misma función que se vio en el widget *Checkbutton*. *Ver aqui*.

flash

invoke

Este método cumple la misma función que se vio en el widget *Checkbutton*. *Ver aqui*.

select

Este método cumple la misma función que se vio en el widget *Checkbutton*. *Ver aqui*.

CAPÍTULO 14

Opciones

Muchos de los widgets que utilizamos y vimos hasta ahora cuentan con opciones, con el fin de no repetir la descripcion de cada uno de ellos seran vistos uno a uno en esta seccion. Esta seccion no es mas que una traduccion de la documentacion oficial de Tk sobre la libreria Tkinter con ejemplos e imagenes.

Por hacer

Enlazar con seccion para los colores

Nota: Para mas informacion sobre los colores visite la seccion "Colores" que se ve en

Pueden encontrar mas informacion desde la documentacion oficial de Tcl/Tk sobre las opciones estandar http://www.tcl.tk/man/tcl8.5/TkCmd/options.htm

Por hacer

Especificar widgets afectados por cada opcion (Ver formato de impresion)

14.1 activebackground

Con esta opcion podemos indicar que color de fondo se debera utilizar cuando el cursor del mouse se posicione sobre el widget. Para algunos elementos y en algunas plataformas solo sera visible cuando se hace click sobre el.

Acepta cualquier color en un formato valido para Tk, ver en seccion "Colores"

```
boton = Tkinter.Button(root, text="Hola Mundo!!!", activebackground="#F50743")
boton.pack()
```

14.2 activeforeground

Al igual que pasa con 'activebackground' con esta otra opcion podemos especificar el color en este caso de primer plano del widget cuando se posiciona el cursor sobre este.

```
boton = tk.Button(root, text="Hola Mundo!!!", activeforeground="#F50743")
```

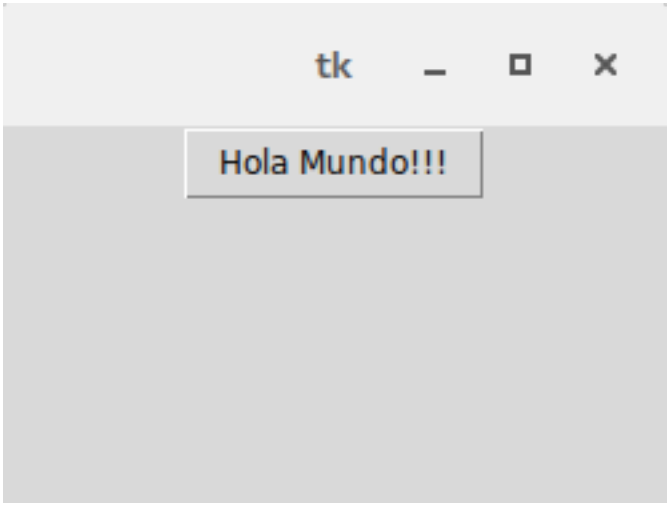


Figura 14.1: Ejemplo sin el cursor encima

Figura 14.2: Ejemplo con el cursor encima

Se muestra con color '#F50743' (Rojo)

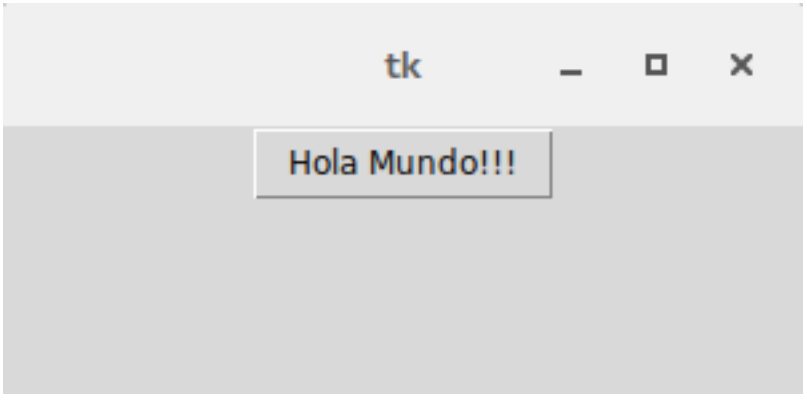


Figura 14.3: Ejemplo sin el cursor encima

Figura 14.4: Ejemplo con el cursor encima
Se muestra con color '#F50743' (Rojo)

14.3 anchor

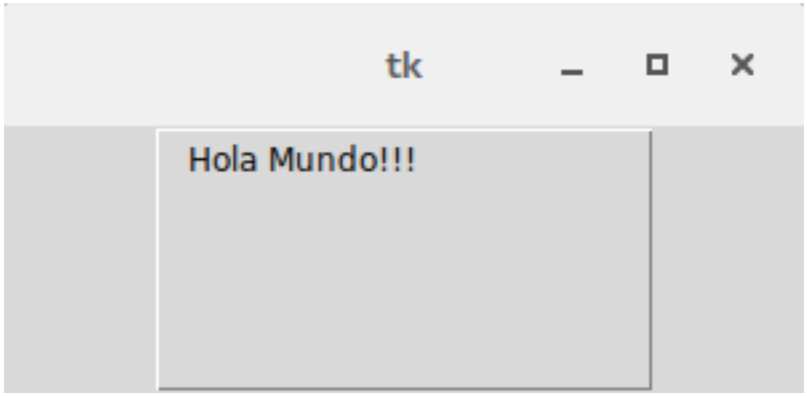
Tkinter nos permite controlar de que forma se va a posicionar un texto o imagen con respecto al widget en el que se encuentra. La opcion por defecto es 'CENTER' que muestra el elemento centrado.

NW	N	NE
W	CENTER	E
SW	S	SE

14.3.1 NW

Posiciona el texto/imagen en la esquina superior izquierda del widget

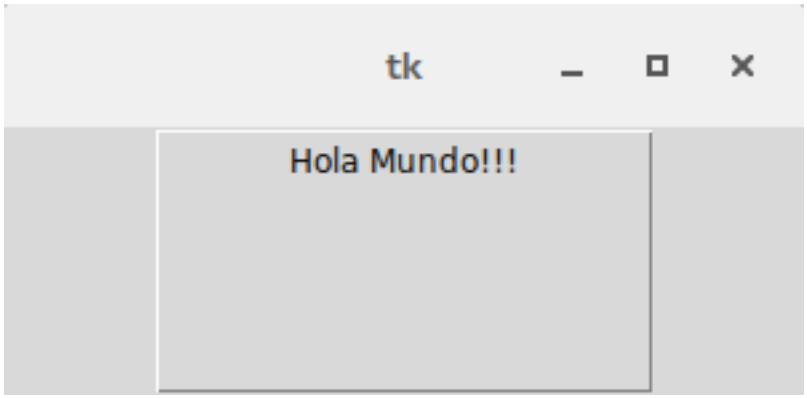
```
boton = tk.Button(root, text="Hola Mundo!!!", width=20, height=10, anchor="nw")
```



14.3.2 N

Posiciona el texto/imagen en la parte superior del widget

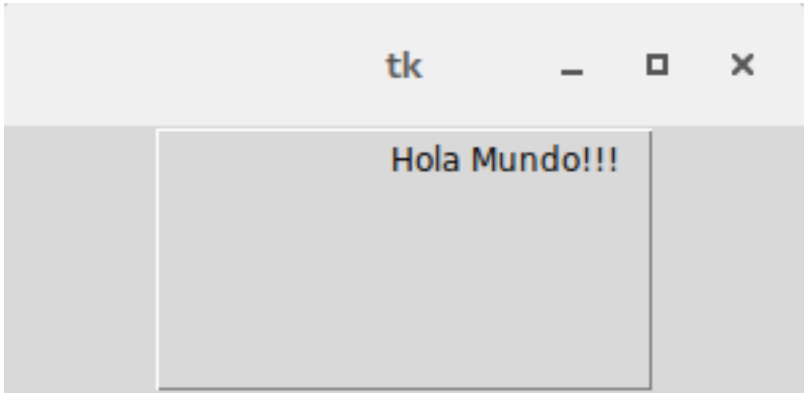
```
boton = tk.Button(root, text="Hola Mundo!!!", width=20, height=10, anchor="n")
```



14.3.3 NE

Posiciona el texto/imagen en la esquina superior derecha del widget

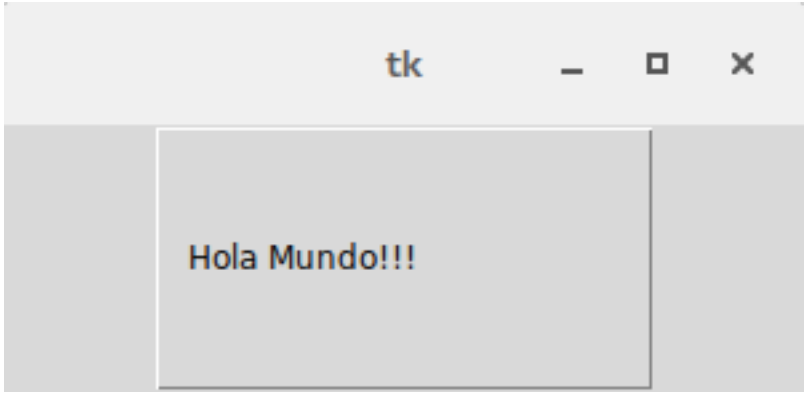
```
boton = tk.Button(root, text="Hola Mundo!!!", width=20, height=10, anchor="ne")
```



14.3.4 W

Posiciona el texto/imagen en la parte izquierda del widget

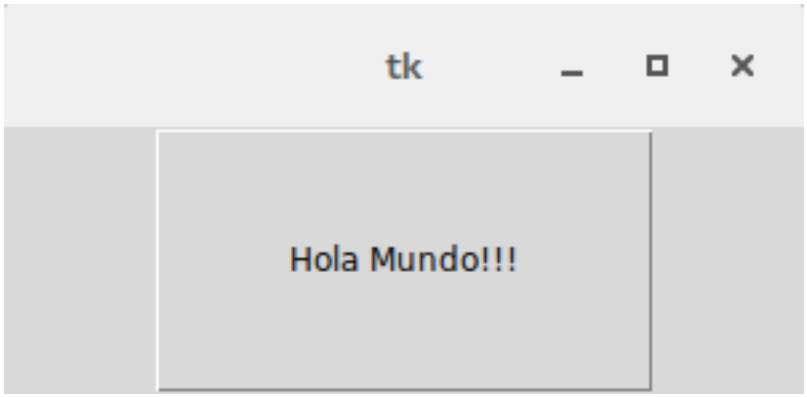
```
boton = tk.Button(root, text="Hola Mundo!!!", width=20, height=10, anchor="w")
```



14.3.5 CENTER

Posiciona el texto/imagen en el centro del widget

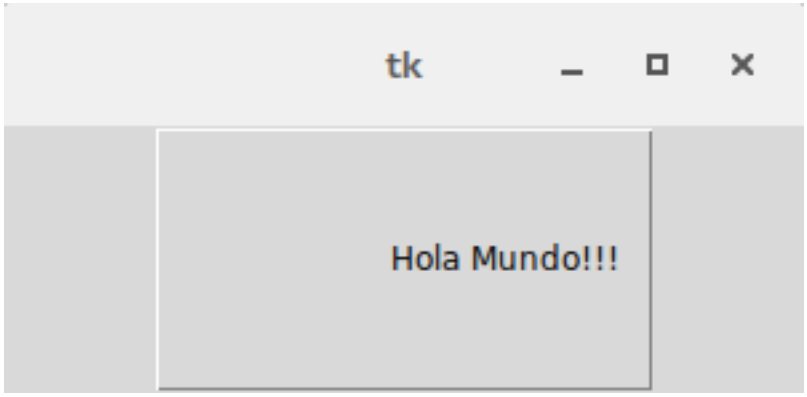
```
boton = tk.Button(root, text="Hola Mundo!!!", width=20, height=10, anchor="center")
```



14.3.6 E

Posiciona el texto/imagen en la parte derecha del widget

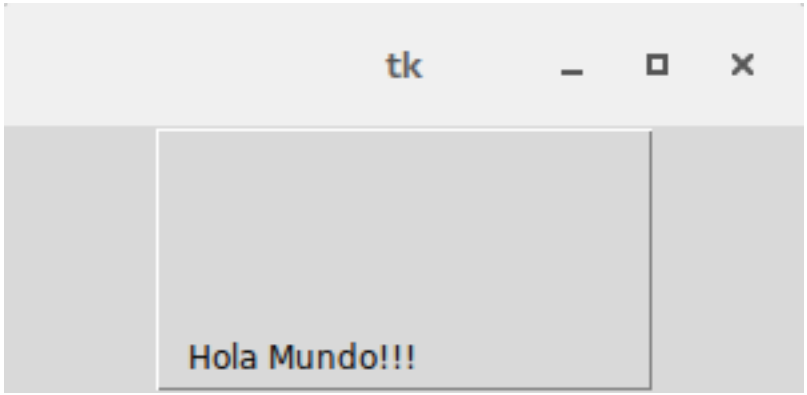
```
boton = tk.Button(root, text="Hola Mundo!!!", width=20, height=10, anchor="e")
```



14.3.7 SW

Posiciona el texto/imagen en la esquina inferior izquierda del widget

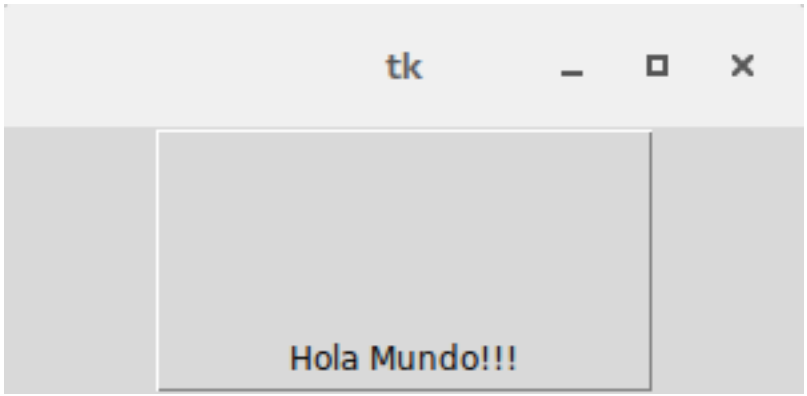
```
boton = tk.Button(root, text="Hola Mundo!!!", width=20, height=10, anchor="sw")
```



14.3.8 S

Posiciona el texto/imagen en la parte inferior del widget

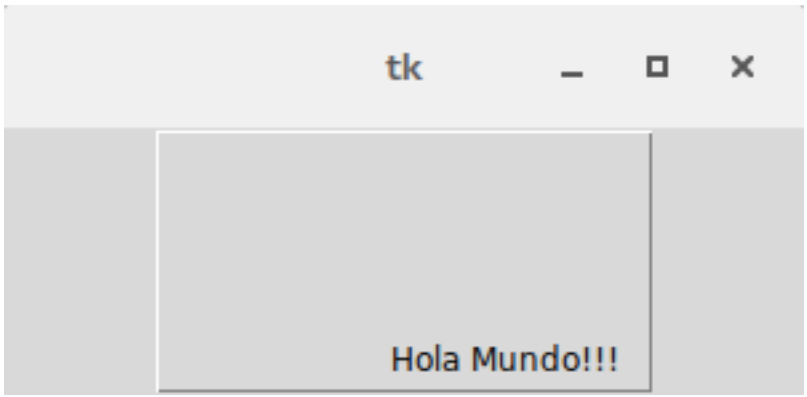
```
boton = tk.Button(root, text="Hola Mundo!!!", width=20, height=10, anchor="s")
```



14.3.9 SE

Posiciona el texto/imagen en la esquina inferior derecha del widget

```
boton = tk.Button(root, text="Hola Mundo!!!", width=20, height=10, anchor="se")
```



14.4 background (bg)

Sirve para indicar el color de fondo que tendra en area del widget cuando este se encuentre inactivo. Tambien se puede utilizar su forma abreviada 'bg' como podemos ver en el segundo ejemplo.

```
boton = tk.Button(root, text="Hola Mundo!!!", background="#38EB5C")
```

```
boton = tk.Button(root, text="Hola Mundo!!!", bg="#38EB5C")
```

Figura 14.5: Vemos el color verde que toma el boton

14.5 bitmap

Por hacer

Ampliar documentacion

```
boton_error = tk.Label(root, bitmap="error").pack()
boton_gray75 = tk.Label(root, bitmap="gray75").pack()
boton_gray50 = tk.Label(root, bitmap="gray50").pack()
boton_gray25 = tk.Label(root, bitmap="gray25").pack()
boton_gray12 = tk.Label(root, bitmap="gray12").pack()
boton_hourglass = tk.Label(root, bitmap="hourglass").pack()
boton_info = tk.Label(root, bitmap="info").pack()
boton_questhead = tk.Label(root, bitmap="questhead").pack()
boton_question = tk.Label(root, bitmap="question").pack()
boton_warning = tk.Label(root, bitmap="warning").pack()
```

14.6 borderwidth (bd)

Nos da la posibilidad de especificar el ancho del borde del elemento con el que estemos trabajando, Tkinter nos permite especificar el valor en varias unidades de medida (centimetros, pulgadas, milimetros y puntos) las cuales seran vistas mas adelante en la guia. Si no se utiliza junto con la opcion 'relief' visualmente es similar a utilizar las opciones 'heigth' y 'width' como se puede observar en el ejemplo. El valor por defecto es de 1 pixel.

```
boton = tk.Button(root, text="Hola Mundo!!!", borderwidth=15)
```

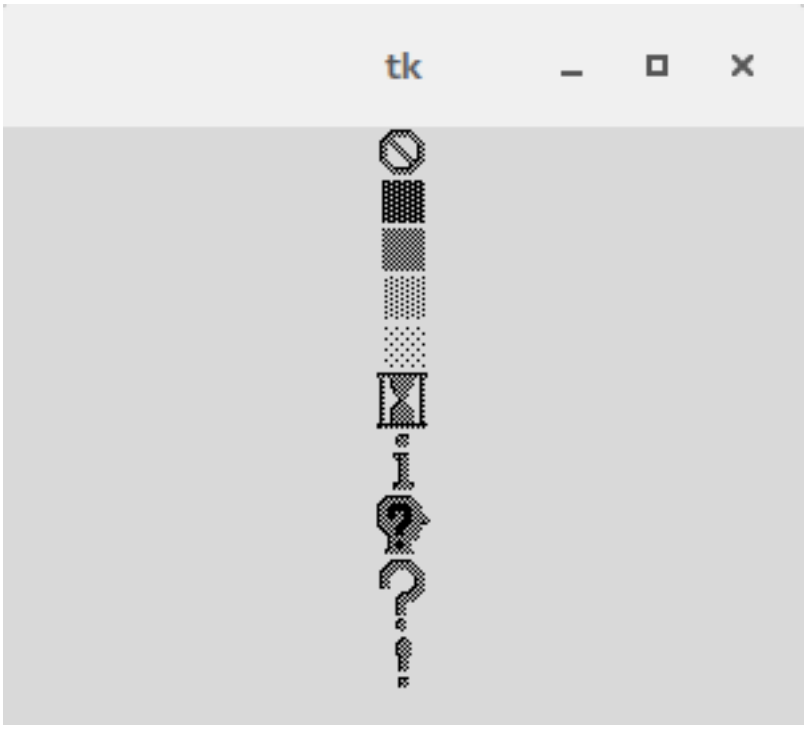


Figura 14.6: Muestra de los bitmaps predefinidos

```
boton = tk.Button(root, text="Hola Mundo!!!", bd=15)
```

Figura 14.7: Un boton con un borde de 15 pixeles

14.7 command

Como ya lo mencionamos cuando vimos al widget Button este permite indicar la funcion o metodo que queremos que se llame cuando se preciona por ejemplo un boton, pero esta opcion tambien funciona con otros widgets de Tkinter cumpliendo la misma funcion.

```
>>> def callback():
>>>     print("Excelente")
```

```
>>> boton = tk.Button(root, text="Que te parece la guia?", command=callback)
'Excelente'
```

14.8 cursor

Por hacer

Crear seccion y enlazar

Nos da la posiblidad de indicar que cursor queremos que se muestre cuando el mouse se posiciona sobre la etiqueta. En el siguiente enlace podremos encontrar una lista con los cursores disponibles en Tkinter http://www.tcl.tk/man/tcl8.5/TkCmd/cursors.htm (Ver seccion "Anexos" para ver el listado completo).

El valor 'none' consigue eliminar el cursor cuando se pasa el cursor sobre el widget.

```
etiqueta = tk.Label(root, text="Que te parece la guia?", cursor="hand1")
```

Figura 14.8: Mostrando una mano a diferencia del cursor normal

14.9 compound

Especifica si el widget debe mostrar texto y mapas/imágenes al mismo tiempo, y si es así, donde el mapa de bits o imagen debe estar en relación con el texto. Debe ser uno de los valores "none", "bottom", "top", "left", "right", o "center". Por ejemplo, el valor (predeterminado) "none" especifica que el mapa de bits o imagen (si se ha definido en el sistema) se mostrará en lugar del texto, el valor "left" especifica que el mapa de bits o imagen se muestra a la izquierda del texto y el valor "center" especifica que el mapa de bits o imagen debe aparecer en la parte superior del texto.

14.10 default

Por hacer

Escribir documentacion (Opcion valida para Button)

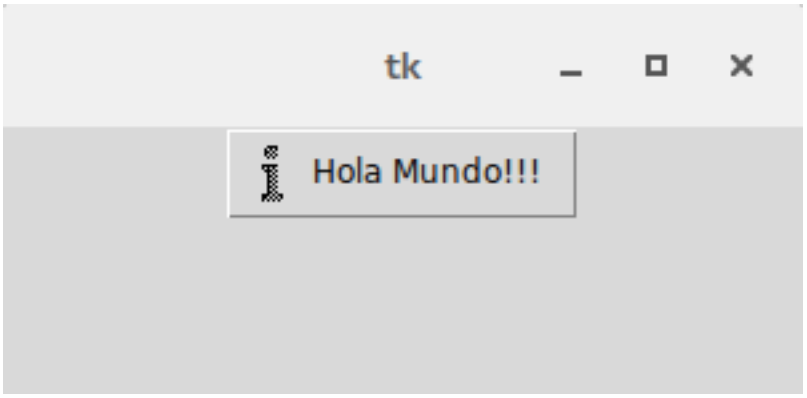


Figura 14.9: Usando compound con el valor left

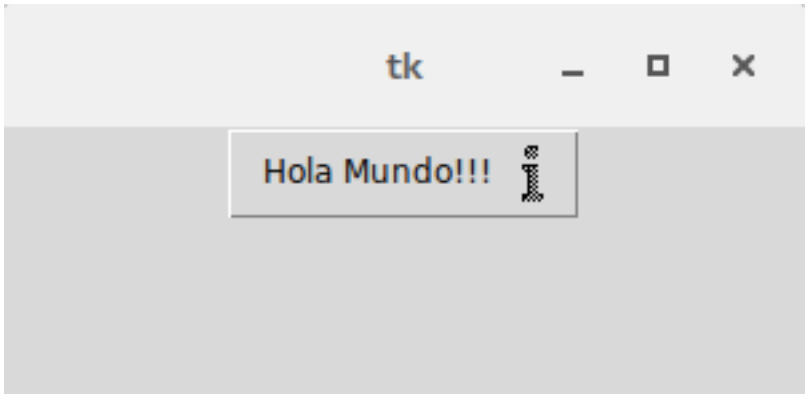


Figura 14.10: Usando compound con el valor right

Figura 14.11: Usando compound con el valor top

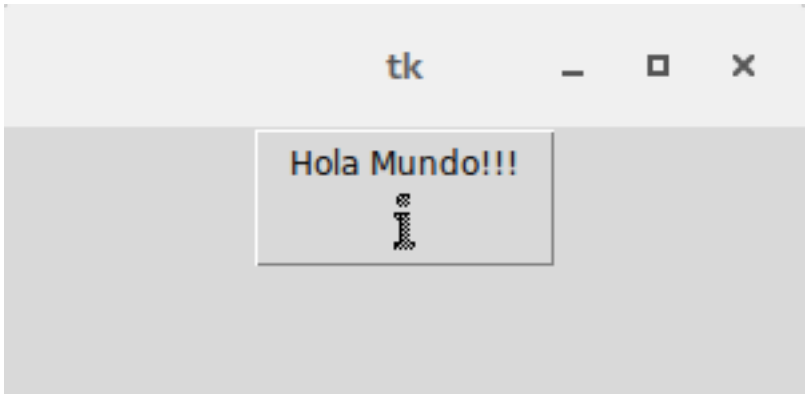


Figura 14.12: Usando compound con el valor bottom

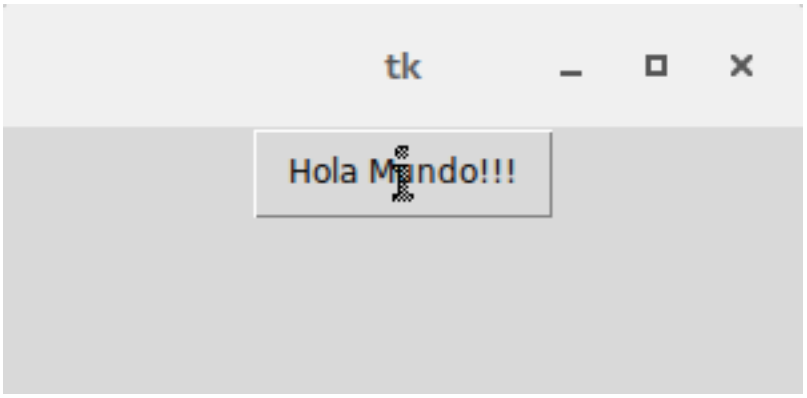


Figura 14.13: Usando compound con el valor center

Figura 14.14: Usando compound con el valor none

14.11 disabledbackground

Por hacer

Escribir documentacion (Opcion valida para Spinbox)

14.12 disabledforeground

Funciona de la misma forma que foreground y activeforeground pero para elementos deshabilitados, se le puede pasar cualquier color en un formato valido, si no se le pasa una cadena vacia el color que se mostrara sera el de la opcion foreground pero con un efecto punteado.

El valor por defecto es *None* y obtiene el color del sistema.

```
boton = tk.Button(root, text="Hola Mundo!!!", state="disabled", disabledforeground=None)
```

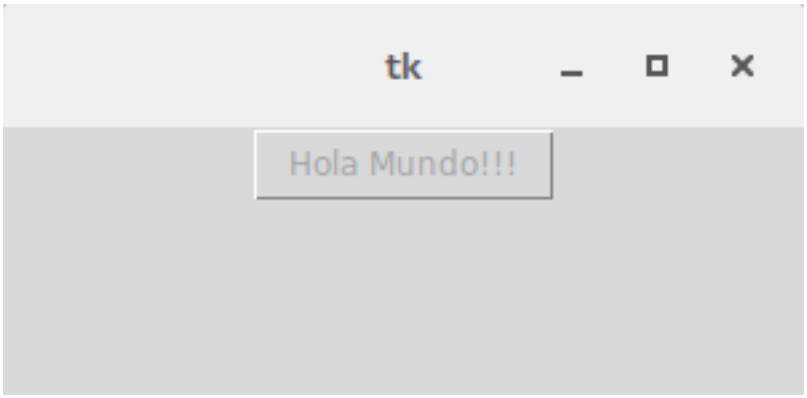


Figura 14.15: Usando disabledforeground con el color por defecto

```
boton = tk.Button(root, text="Hola Mundo!!!", state="disabled", disabledforeground="#F50743")
```

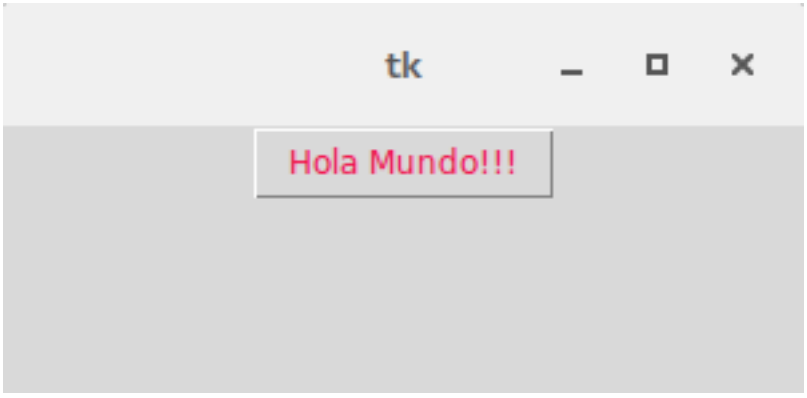


Figura 14.16: Usando disabledforeground con un color rojo

```
boton = tk.Button(root, text="Hola Mundo!!!", state="disabled", disabledforeground="")
```

Figura 14.17: Usando disabledforeground sin darle ningun valor

14.13 exportselection

14.14 font

14.15 foreground (fg)

Por hacer

- Enlazar Label y Entry con sus secciones
- Ver ejemplos para widgets especificos
- Mejorar descripcion

Si deseamos cambiar el color de primer plano usamos foreground, para explicarlo un poco mejor asi como vimos background para determinar el color de fondo del widget tambien podemos cambiar el color de la parte delantera. Por lo general por ejemplo en Label y Entry el color de foreground modifica el color de la fuente. Este color es mostrado en el estado normal del widget, completando el circulo de estados ya vistos (normal, activo y desactivado).

Permite usar la abreviatura **fg** como habiamos visto para *background*.

```
boton = Tkinter.Button(root, text="Hola Mundo!!!", command=funcion, foreground="#38EB5C")
```

```
boton = Tkinter.Button(root, text="Hola Mundo!!!", command=funcion, fg="#38EB5C")
```

14.16 height

Por hacer

Opcion valida para Button

Permite indicar la altura a la que se posicionara nuestro widget en lineas no en pixeles. Si esta opcion no se especifica el tamaño del elemento se ajustara dependiendo el contenido del mismo.

Con None especificamos su valor por defecto que es 0.

Figura 14.18: Mostrando una fuente verde con foreground

```
boton = tk.Button(root, text="Hola Mundo!!!", height=0)
```

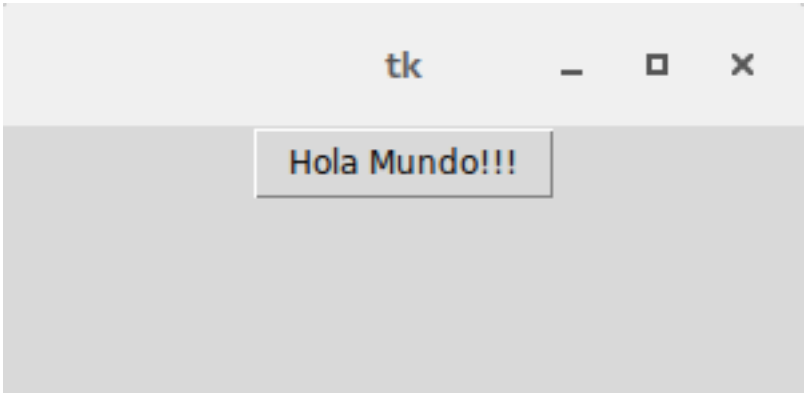


Figura 14.19: Mostrando una altura de 0

```
boton = tk.Button(root, text="Hola Mundo!!!", height=5)
```

```
boton = tk.Button(root, text="Hola Mundo!!!", height=10)
```

14.17 justify

El valor por defecto es center.

14.17.1 RIGHT

```
boton = tk.Button(root, text="Hola\nMundo!!!", justify="right")
```

14.17.2 CENTER

```
boton = tk.Button(root, text="Hola\nMundo!!!", justify="center")
```

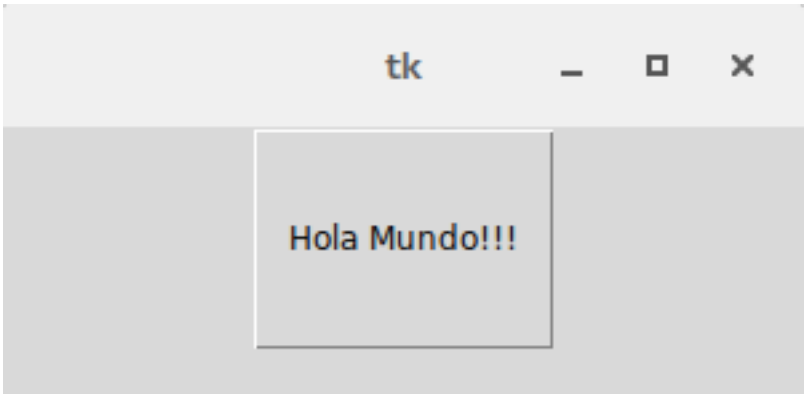


Figura 14.20: Mostrando una altura de 5

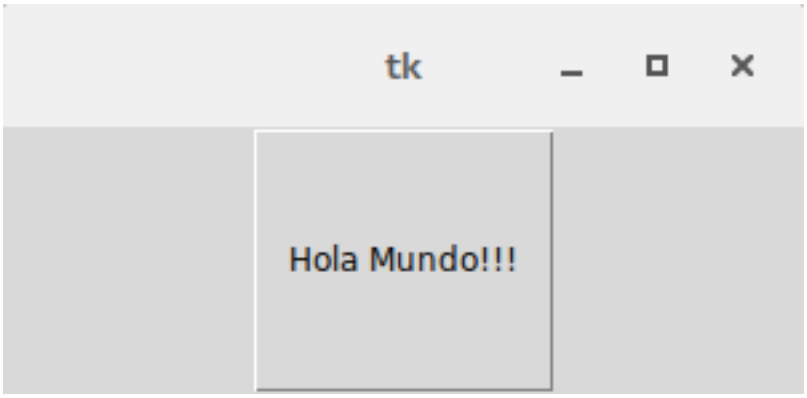


Figura 14.21: Mostrando una altura de 10

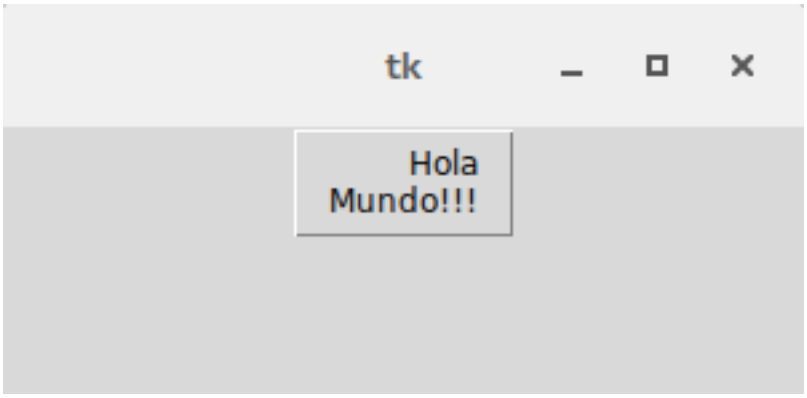


Figura 14.22: Orientando el texto a la derecha

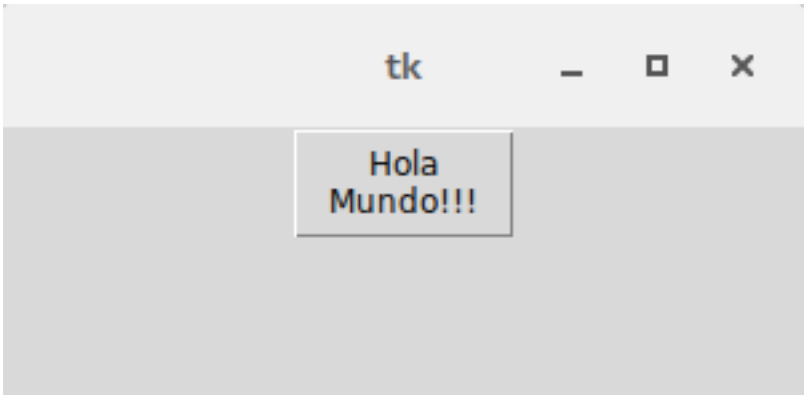


Figura 14.23: Centrando el texto

14.17.3 LEFT

```
boton = tk.Button(root, text="Hola\nMundo!!!", justify="left")
```

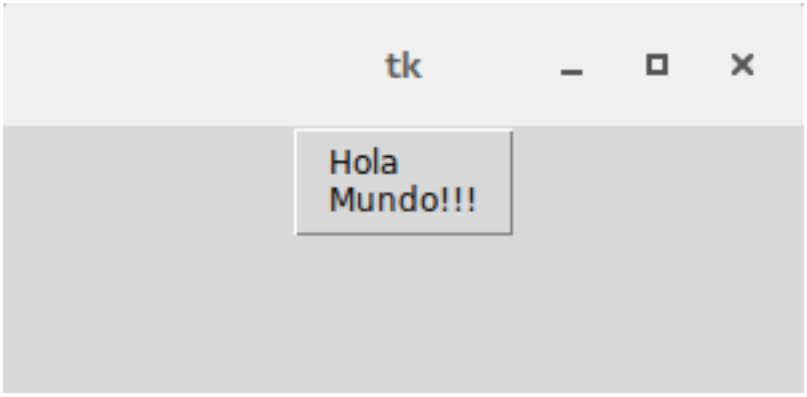


Figura 14.24: Orientando el texto a la izquierda

14.18 overrelief

Por hacer

Escribir documentacion (Se aplica a Button)

Especifica un relieve alternativo para el botón, que se utilizará cuando el cursor del ratón este sobre el widget. Esta opcion puede ser usada para hacer toolbar buttons, configurando relief="flat" y overrelief="raised".

La cadena vacía es el valor predeterminado.

```
boton = tk.Button(root, text="Hola Mundo!!!", overrelief="")
```

```
boton = tk.Button(root, text="Hola Mundo!!!", overrelief="flat")
```

```
boton = tk.Button(root, text="Hola Mundo!!!", overrelief="raised")
```

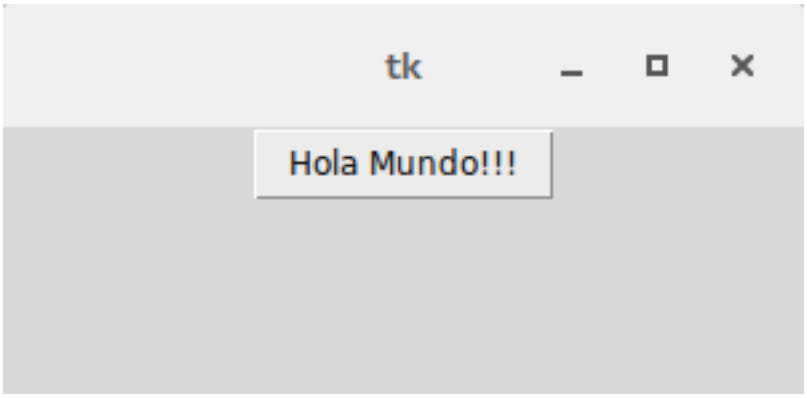


Figura 14.25: overrelief con valor None

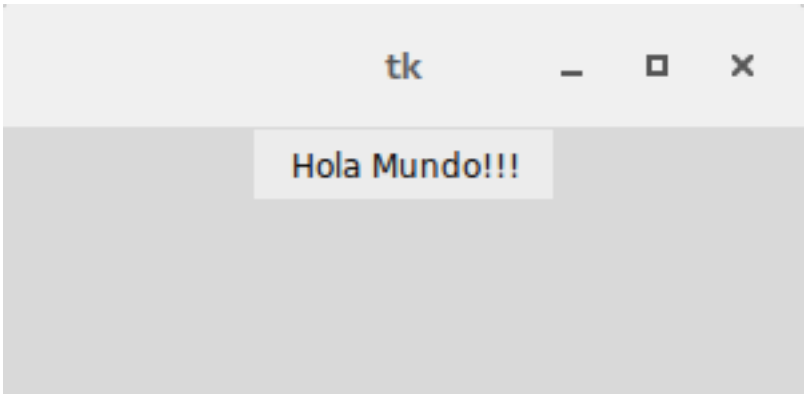


Figura 14.26: overrelief con valor

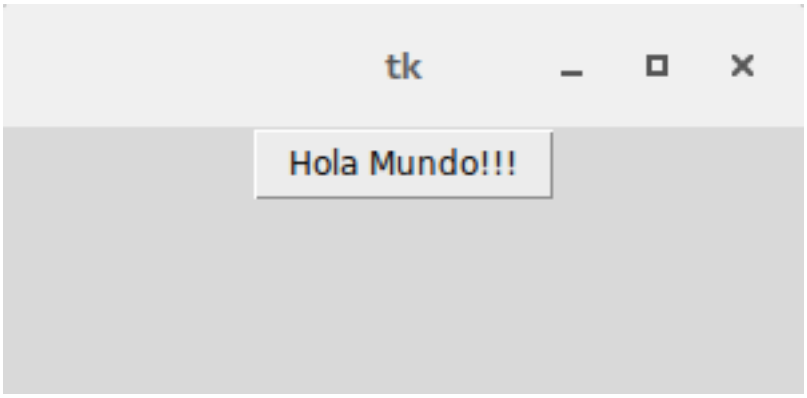


Figura 14.27: overrelief con valor

```
boton = tk.Button(root, text="Hola Mundo!!!", relief="flat", overrelief="raised")
```

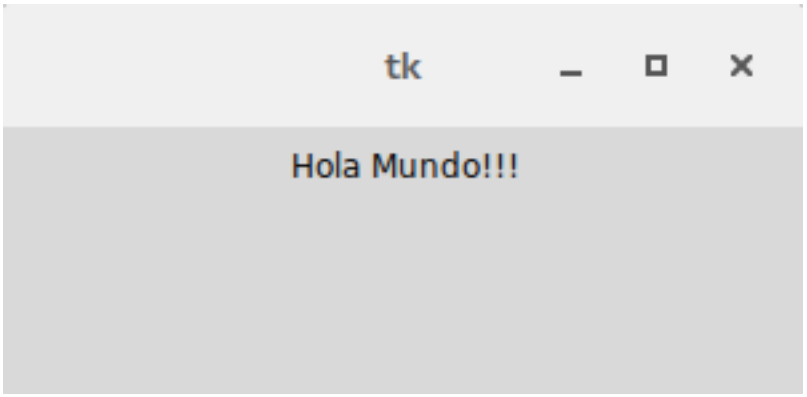


Figura 14.28: Estilo flat-raised sin cursor encima

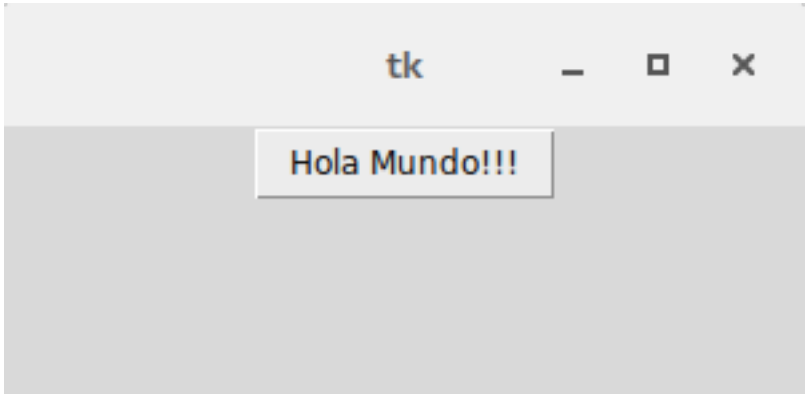


Figura 14.29: Estilo flat-raised con cursor encima

14.19 relief

Por hacer

Profundizar introduccion

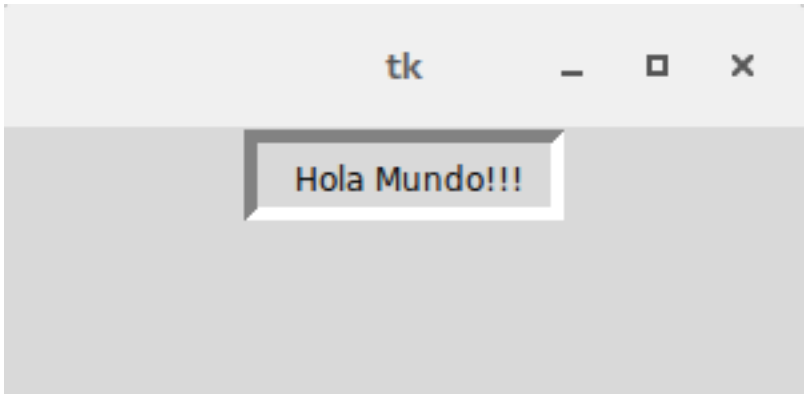
Especifica la apariencia de un borde decorativo alrededor del widget que simula un efecto 3D sobre el elemento, el ancho de esta sombra depende del valor que se especifique con el atributo 'borderwidth', los posibles valores son SUNKEN, RAISED, GROOVE, RIDGE, FLAT. El valor por defecto es FLAT que no coloca ningun borde sobre el widget.

Descripcion de los cuatro estilos:

14.19.1 SUNKEN

Borde hundido, que provoca que el elemento que encierra parezca que se encuentra por debajo del nivel de la superficie de la pantalla.

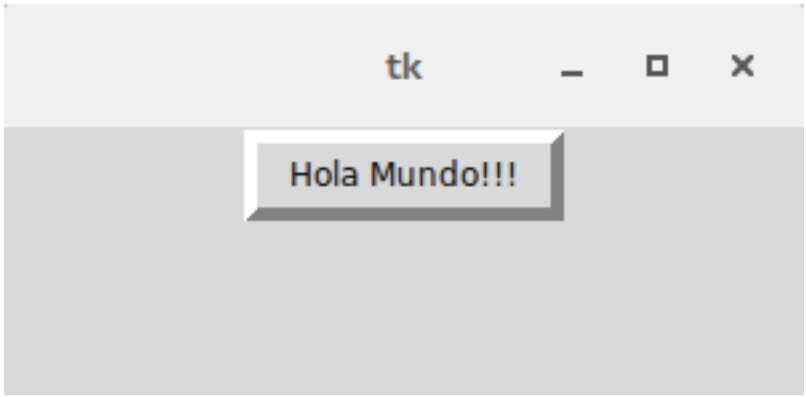
```
boton = tk.Button(root, text="Hola Mundo!!!", relief="sunken", borderwidth=5)
```



14.19.2 RAISED

Borde saliente, que provoca que el elemento que encierra parezca que se encuentra por encima del nivel de la superficie de la pantalla.

```
boton = tk.Button(root, text="Hola Mundo!!!", relief="raised", borderwidth=5)
```



14.19.3 GROOVE

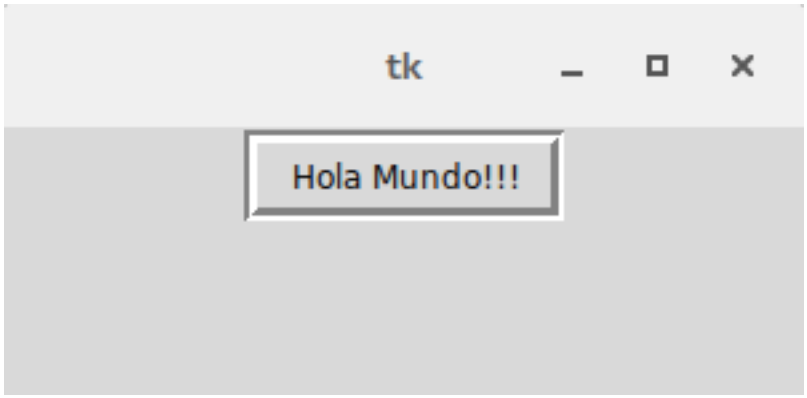
Borde hundido, que visualmente parece que se encuentra por debajo del nivel de la superficie de la pantalla.

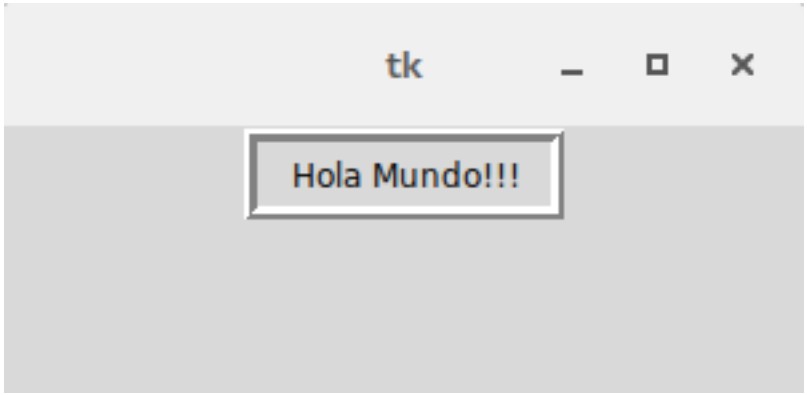
```
boton = tk.Button(root, text="Hola Mundo!!!", relief="groove", borderwidth=5)
```

14.19.4 RIDGE

Borde saliente, que visualmente parece que se encuentra por encima del nivel de la superficie de la pantalla.

```
boton = tk.Button(root, text="Hola Mundo!!!", relief="ridge", borderwidth=5)
```

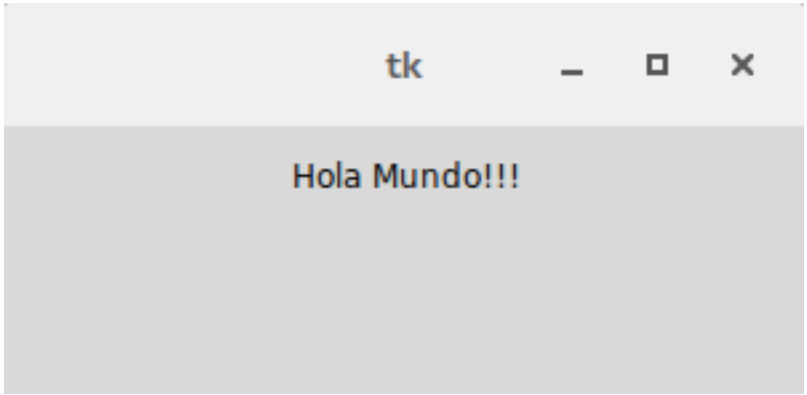




14.19.5 FLAT

No se muestra ningún borde.

```
boton = tk.Button(root, text="Hola Mundo!!!", relief="flat", borderwidth=5)
```



14.20 state

Por hacer

Enlazar opciones 'foreground' y 'background'

Esta opcion especifica tres estados 'normal', 'active' o 'disabled'. En estado normal se muestra el widget usando las opciones 'foreground' y 'background', en estado 'active' se muestra usando las opciones 'activeforeground' y 'activebackground' y por ultimo el estado 'disabled' es mostrado usando 'disabledforeground' y 'background' y deshabilitando el uso del widget.

En el siguiente ejemplo se definen tres botones, las tres tienen los mismos argumentos con lo mismos valores a excepcion de la opcion 'state', en el resultado final podremos ver que al cambiar el valor de 'state' solo toma los valores que habiamos mencionado en la descripcion de la opcion ignorando el resto.

```
from six.moves import tkinter as tk

root = tk.Tk()
root.geometry("300x100")

foreground = "red"
background = "blue"
activeforeground = "green"
activebackground = "black"
disabledforeground = "white"

boton_normal = tk.Button(root, text="normal", state="normal",
                         foreground=foreground,
                         background=background,
                         activeforeground=activeforeground,
                         activebackground=activebackground,
                         disabledforeground=disabledforeground).pack()


boton_active = tk.Button(root, text="active", state="active",
                         foreground=foreground,
                         background=background,
                         activeforeground=activeforeground,
                         activebackground=activebackground,
                         disabledforeground=disabledforeground).pack()

boton_disabled = tk.Button(root, text="disabled", state="disabled",
                         foreground=foreground,
                         background=background,
                         activeforeground=activeforeground,
                         activebackground=activebackground,
                         disabledforeground=disabledforeground).pack()
```

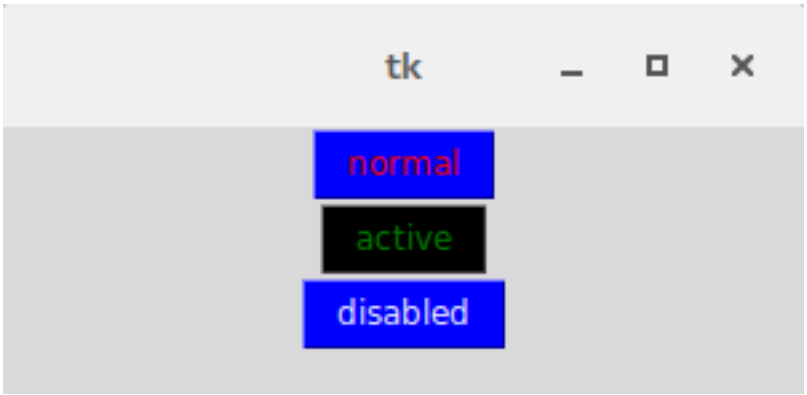


Figura 14.30: Ejemplo de los 3 estados sin el cursor encima

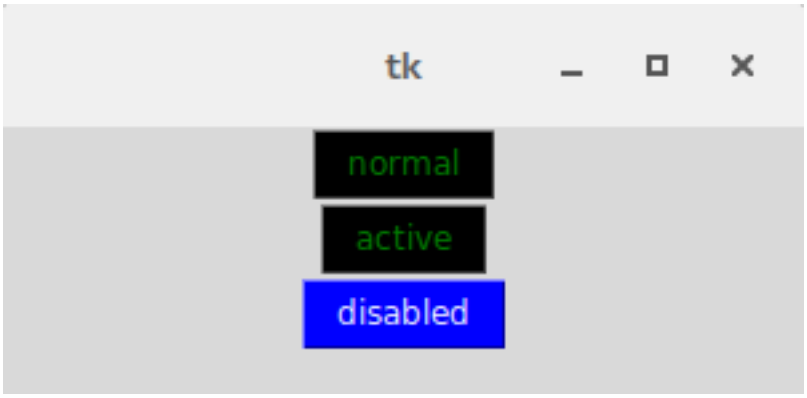


Figura 14.31: Ejemplo con el cursor encima del primer boton (Estado normal)

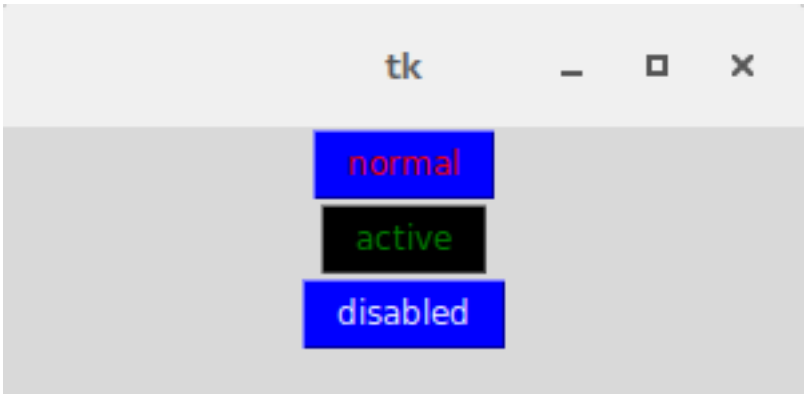


Figura 14.32: Ejemplo con el cursor encima del segundo boton (Estado active)

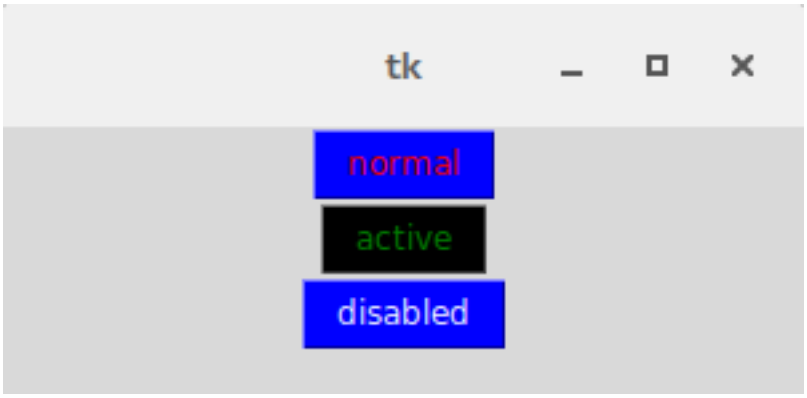


Figura 14.33: Ejemplo con el cursor encima del tercer boton (Estado disabled)

14.21 text

Por hacer

Enlazar opcion compound

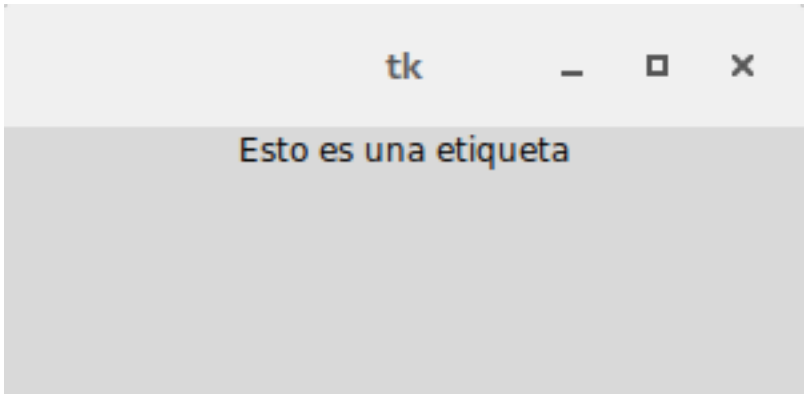
Este es un elemento muy importante para los widgets como Label o Button entre otros, esta opcion nos da la posibilidad de incluir un texto dentro del widget, este texto puede contener saltos de lineas en caso de ser necesario y tambien se puede utilizar una variable para indicar la cadena de texto a usar. Si se utiliza una imagen este elemento es ignorado por Tkinter salvo que usemos la opcion compound.

Ejemplo simple:

```
etiqueta = tk.Label(root, text="Esto es una etiqueta")
```

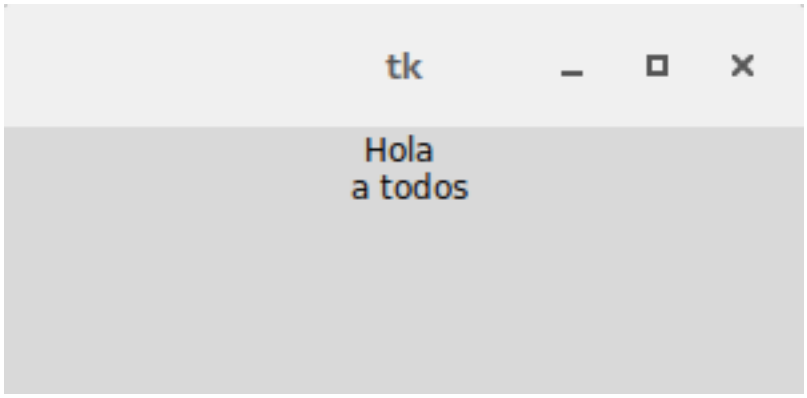
Ejemplo con una variable:

```
texto = "Esto es una etiqueta"
etiqueta = tk.Label(root, text=texto)
etiqueta.pack()
```



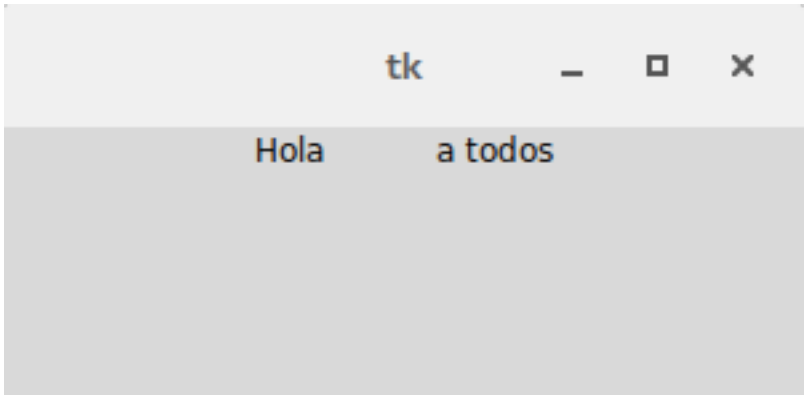
Ejemplo con salto de linea (recuerden que deben usar el caracter n para el salto de linea):

```
etiqueta = tk.Label(root, text="Hola \n a todos")
```



Ejemplo con tabulacion (recuerden que deben usar el caracter t para la tabulacion):

```
etiqueta = tk.Label(root, text="Hola \t a todos")
```


tk
Hola
a todos

CAPÍTULO 15

Colores

Todos los widgets estándar de Tkinter proporcionan un conjunto básico de opciones para poder personalizarlos, que nos permiten modificar sus colores, fuentes, bordes y tamaños entre otros. La mayoría de ellos permite especificar un color de fondo y de primer plano que nos serán muy útiles cuando intentemos darle un aspecto mas pulido a nuestra aplicación.

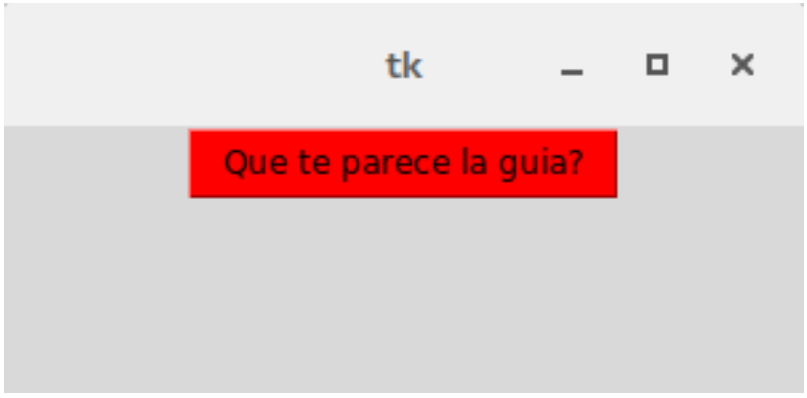
Para especificar dichos colores tenemos dos formas, una es utilizar el nombre del color y la otra consiste en especificar la proporción de rojo, verde y azul que componen a dicho color en dígitos hexadecimales que es la forma mas popular y la mas precisa a la hora de trabajar.

15.1 Nombres de colores

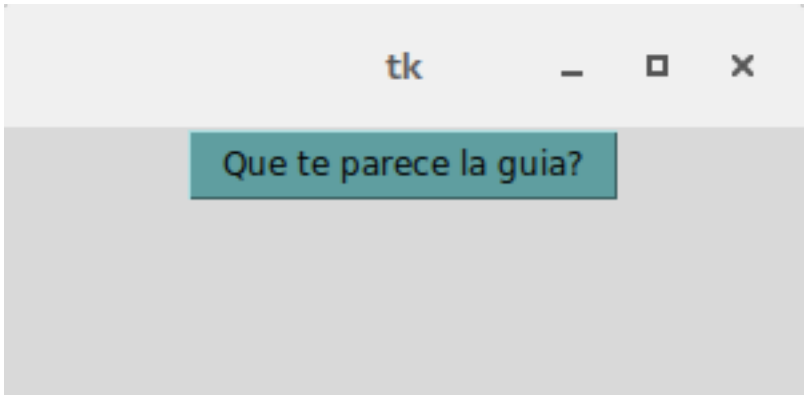
Como se había mencionado se puede especificar un color con solo usar su nombre en ingles, por ejemplo para el color blanco usaríamos la palabra 'white' y para el color rojo 'red', aunque Tkinter no nos limita a usar solamente los colores básicos sino que nos ofrece una paleta bastante amplia de colores.

Les dejo dos ejemplos para que puedan observarlo en funcionamiento:

```
boton = tk.Button(root, text="Que te parece la guia?", background="red")
```




```
boton = tk.Button(root, text="Que te parece la guia?", background="CadetBlue")
```



Otra de las posibilidades que tenemos es utilizar los colores del sistema para conseguir un aspecto mas acorde a la plataforma con que estemos trabajando.

Podemos obtener una lista de los posibles colores que nos da Tcl/Tk en la siguiente pagina [http://www.tcl.tk/man/tcl8.5/TkCmd/colors.htm] o también en los anexos de esta guía los podrán encontrar:

15.2 RGB Hexadecimal

Aunque este método sea un poco mas complicado en su uso, es mas preciso que el método que habíamos mencionado arriba. La forma en que se utiliza es la siguiente:

#RRGGBB

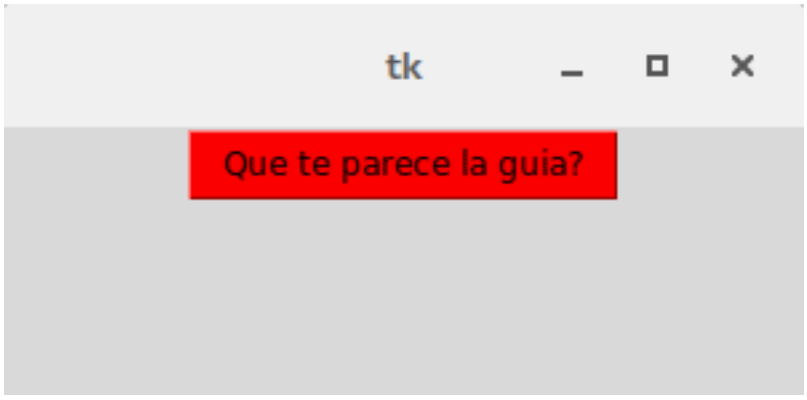
Donde RR, GG y BB son representaciones hexadecimales de los colores rojo, verde y azul, respectivamente. Para formar un color debemos en primer lugar elegir la intensidad que deseamos usar, especificando la cantidad de rojo, verde y azul en una escala del 0 al 255.

Usando la siguiente linea desde la consola de Python podemos crear nuestros colores para Tkinter reemplazando los valores de la tupla por los que deseamos utilizar:

```
color_rojo = "#%02x%02x%02x" % (255, 0, 0)
```

Su uso es muy simple, solo debemos usar la forma que habíamos visto cuando coloreamos usando los nombres pero ahora deberemos reemplazar el nombre del color por su equivalente en el valor RBG Hexadecimal como en el siguiente ejemplo:

```
boton = Tkinter.Button(root, text="Que te parece la guia?", background="#FA0000")
```



Donde “#FA0000” es el equivalente a utilizar la cadena ‘red’ para indicar el nombre del color.

Si estan pensando, me recomiendan los colores hexadecimales pero como hago para saber las proporciones de colores a utilizar, es facil existen innumerables paletas de colores en la web. Un buen recurso para comenzar es la Wikipedia

CAPÍTULO 16

Indices y tablas

- genindex
- search

C

D

G

I

S

T

X