# INTRO

https://www.youtube.com/user/johnphilipjones/featured

https://www.youtube.com/watch?v=yuoSKkSEhQg&list=PL6lxxT7IdTxGoHfouzEK-dFcwr\_QClME\_

from tkinter import \*

my\_window = Tk()

print(id(my\_window ), type(my\_window ))

# Titulo

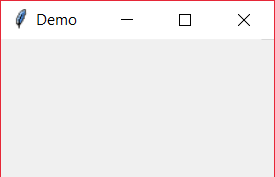
my\_window = Tk()

**my\_window.title("Demo")** # objeto.metodo(atributo)

my\_window.mainloop()

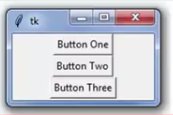
# objetos

Al crear nuestro my\_window, además de tener un objeto en el espacio de ejecución, obtenemos una representación visual.



# Tkinter events and mainloop()

Botones, Pestañas, etc. de nuestra GUI están esperando continuamente cambios en el ratón o teclado para actuar en lo que representa y cómo se representa mediante el código que maneja el evento.

< insertar imagen de ventana con 3 botones >

Nuestra ventana estará continuamente en un bucle esperando eventos. El bucle se ejecuta con el siguiente comando:

my\_window.mainloop()

Cuando salta un evento se resuelve y vuelve al bucle principal.

# How to set the background colour of a Python tkinter window

A continuación vamos a establecer el color de fondo de nuestra ventana con este simple código:

from tkinter import \*  
  
my\_window = Tk()

# CODE GOES HERE  
my\_window.title("Demo")  
  
# set the background of the window  
# Se puede usar bg en vez de background

# Se puede usar número hexadecimal: '#00ff00' en vez de 'green'  
**my\_window.configure(background='red')**   
  
#bucle principal  
my\_window.mainloop()

Obteniendo algo así:

Ejemplos de colores en hexadecimal:

<https://www.w3schools.com/colors/colors_picker.asp>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hexadecimal | Red | Green | Blue | Color |
| #00ff00 | 00 | ff | 00 |  |
| #ff7700 | ff | 77 | 00 |  |
| #88324f | 88 | 32 | 4f |  |
| #00ffbf | 00 | ff | bf |  |

# Setting the Size of a tkinter Window

Para definir más parámetros: **altura** (height) y **ancho** (width) de nuestra ventana se utiliza también el método config, esta vez con más variables.

my\_window.configure(width=400, height=200, background='#94d42b')

En este caso definimos:

* ancho = 400px
* altura = 200px
* color = #94d42b

# How to set the location of a Python tkinter window

Para este caso usamos el método geometry. Con el siguiente código conseguimos lo mismo que con configure:

# my\_window.configure(width=400, height=200)

my\_window.geometry("400x200")

Por defecto la ventana se nos muestra cerca del borde superior izquierdo de la pantalla. Si queremos una posición más específica debemos incluir lo siguiente:

# Justo en el borde superior izquierdo: x=0, y=0

my\_window.geometry("400x200+0+0")

# x=500 -> 500px desde el borde izquierdo

# y=50 -> 50px desde el borde superior de la pantalla

my\_window.geometry("400x200+500+50")

# my\_window.geometry('width'x'heith'+'x\_position'+'y\_position')

# Creating a fixed size Python tkinter window

Algunas de las modificaciones que podemos hacer a nuestra ventana una vez que la estamos visualizando en la pantalla son: modificar tamaño, minimizar, maximizar, cerrar. Con el siguiente código definimos cuales de ellas son modificables:

# sólo permite cambio de tamaño de altura, no de ancho

my\_window.resizable(width=False, height=True)

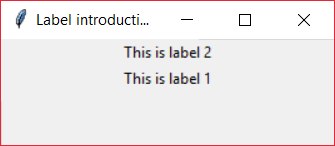
# How to place a Python tkinter window in the center of the screen

Con el siguiente código calculamos internamente (x, y) que son las coordenadas NW de nuestra ventana

from tkinter import \*  
  
my\_window = Tk() # crea un objeto por defecto  
my\_window.title("Ventana en centro de pantalla")  
  
# window size  
width\_of\_window = 400  
heigth\_of\_window = 200  
  
# screen size  
screen\_width = my\_window.winfo\_screenwidth()  
screen\_heigth = my\_window.winfo\_screenheight()  
  
# coordinate por (x,y) NW window position  
x\_coordinate = int((screen\_width/2) - (width\_of\_window/2))  
y\_coordinate = int((screen\_heigth/2) - (heigth\_of\_window/2))  
  
# geometry  
**my\_window.geometry('{}x{}+{}+{}'.format(width\_of\_window,  
 heigth\_of\_window,  
 x\_coordinate,  
 y\_coordinate))**  
# bucle principal  
my\_window.mainloop()

# Introduction to the tkinter Label widget

Para obtener una ventana que nos muestre la siguiente visualización debemos usar el código que se incluye. **El orden de los label dependen del orden en que se emplee pack().**



my\_window = Tk() # crea un objeto por defecto  
  
my\_window.title("Label introduction")  
  
# Creamos label en nuestra ventana  
**label\_1 = Label(my\_window, text='This is label 1')  
label\_2 = Label(my\_window, text='This is label 2')**  
  
# el orden en que aparecen labels en la ventana dependen del orden en que usemos pack()  
**label\_2.pack()**

**label\_1.pack()**  
  
*# bucle principal*my\_window.mainloop()

# Setting the font type, font colour and font size of a label

label\_1 = Label(my\_window, text='This is label 1', background="blue", foreground="white")  
label\_2 = Label(my\_window, text='This is label 2', bg='#7fff00', fg='black')

label\_3 = Label(my\_window,  
 text='Blue text in Verdana 16 bolt italic Font',  
 fg=**'blue'**,  
 font=**'Verdana 16 bold italic'**)

# el orden en que aparecen labels en la ventana dependen del orden en # que usemos pack()  
label\_1.pack()

label\_2.pack()  
label\_3.pack()



# Setting the width of a tkinter label

El ancho del que hablaremos a continuación es llamado 'text units'

label\_1 = Label(my\_window, text='This is label 1', background="red", font='Times 12', **width=20**)  
label\_2 = Label(my\_window, text='This is label 2', bg='#7fff00', font='Times 12', **width=30**)  
label\_3 = Label(my\_window, text='This is label 3', bg='blue', font='Times 12', **width=40**)

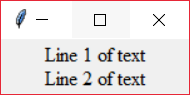
Obtenemos algo tal que así:

El ancho de cada **label va en función del tamaño de la fuente**, en este caso 12. Si la fuente fuese mayor, la anchura sería proporcionalmente mayor.

# Displaying multiple lines of text in a Python tkinter label

En una misma Label podemos utilizar 2 líneas con un simple retorno de carro '\n' como en el código que se muestra a continuación:

label\_1 = Label(my\_window, text=**'Line 1 of text\nLine 2 of text'**, font='Times 12')  
label\_1.pack()



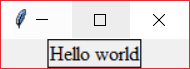
# Setting the look of the border of a Python tkinter label

Adicionalmente se puede usar **borderwidth = bd**.

Además algunas de las opciones posibles para relief: **solid, raised, sunken, ridge, groove, flat.**

label\_1 = Label(my\_window, text='Hello world', borderwidth=1, relief='solid', font='Times 12')

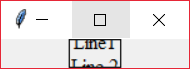
Recordamos que se puede utilizar background = bg, foreground = fg, borderwidth = bd



# Setting the height of a Python tkinter label

Al utilizar el atributo **height=N**, definimos el número de líneas de texto que tendrá la altura de nuestro label. Con el siguiente ejemplo:

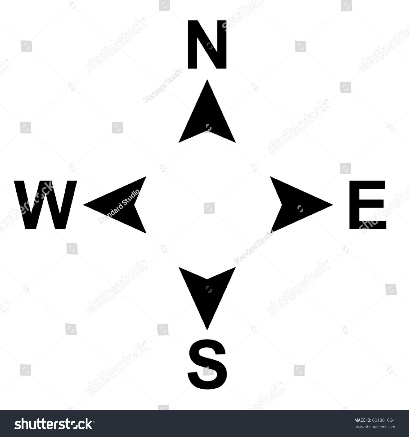
label\_1 = Label(my\_window, text=**'Line1\nLine 2'**,   
 borderwidth=1,   
 relief='solid',   
 font='Times 12',   
 **height=1**)  
label\_1.pack()

Obtenemos la siguiente ventana:

Esto queda solucionado utilizando **height=2,** que son el número de líneas de texto que queremos en nuestra visualización.

Nótese que debido al número de parámetros empieza a hacer ilegible el código, ponemos cada uno de ellos en una línea diferente.

# Positioning text within a Python tkinter label

Se usa las siguientes letras para el posicionamiento:

* N = North
* S = South
* W = West
* E = East

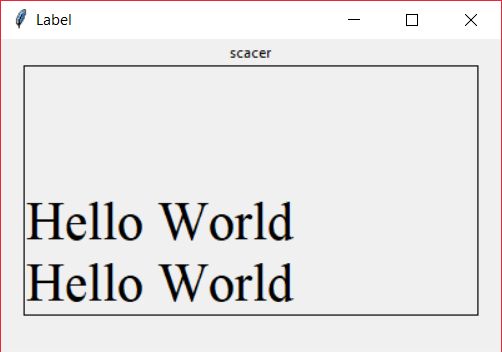
En el momento de la creación del Label hay que pasarle un anchor de la lista que mostramos:

* **CENTER, N, NE, E, SE, S, SW, W, NW**

**Por defecto** si no se especifica una de las direcciones mencionadas antes, el Label se posiciona el en **centro**.

Veamos que ocurre con el siguiente código:

label\_1 = Label(my\_window,  
 text='scacer')  
label\_2 = Label(my\_window,  
 text='Hello World\nHello World',  
 borderwidth=1,  
 relief='solid',  
 font='Times 32',  
 width=15,  
 height=4,  
 **anchor=SW**)  
label\_1.pack()  
label\_2.pack()



# How to pad space around the text of Python tkinter label

label\_1 = Label(my\_window,  
 text='spacer')  
label\_2 = Label(my\_window,  
 text='Hello World',  
 borderwidth=1,  
 relief='solid',  
 font='Times 32')  
label\_3 = Label(my\_window,  
 text='spacer')  
label\_4 = Label(my\_window,  
 text='Hello World',  
 borderwidth=1,  
 relief='solid',  
 font='Times 32',  
 **padx=80**, # padx añade padding horizontal a ambos lados  
 **pady=20**) # pady similar pero vertical  
  
label\_1.pack()  
label\_2.pack()  
label\_3.pack()  
label\_4.pack()



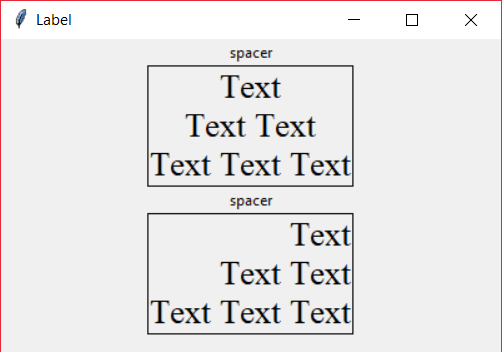
# How to justify text in a Python tkinter label

En el momento de la creación del Label hay que pasarle el tipo de justificación que queremos de la lista que mostramos. Si no especificamos nada estará centrado:

* **CENTER, RIGHT, LEFT**

label\_1 = Label(my\_window,  
 text='spacer')  
label\_2 = Label(my\_window,  
 text='Text\nText Text\nText Text',  
 borderwidth=1,  
 relief='solid',  
 font='Times 20')  
label\_3 = Label(my\_window,  
 text='spacer')  
label\_4 = Label(my\_window,  
 text='Text\nText Text\nText Text',  
 borderwidth=1,  
 relief='solid',

font='Times 20',   
 **justify=RIGHT**)

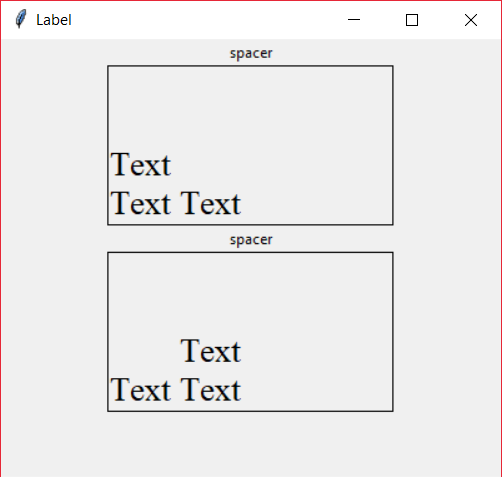


# How to anchor and justify text in a Python tkinter label

Podemos combinar según la necesidad anchor y justify:

label\_1 = Label(my\_window,  
 text='spacer')  
label\_2 = Label(my\_window,  
 text='Text\nText Text',  
 borderwidth=1,  
 relief='solid',  
 font='Times 20',  
 width=15,  
 height=4,  
 **anchor=SW,  
 justify=LEFT**)  
label\_3 = Label(my\_window,  
 text='spacer')  
label\_4 = Label(my\_window,  
 text='Text\nText Text',  
 borderwidth=1,  
 relief='solid',  
 font='Times 20',  
 width=15,  
 height=4,  
 **anchor=SW,  
 justify=RIGHT**)  
  
label\_1.pack()  
label\_2.pack()  
label\_3.pack()  
label\_4.pack()

Obtenemos la siguiente ventana, el label está situado en el SW, pero la justificación es diferente en cada caso.



# How to access the options of a Python tkinter label

label\_1 = Label(my\_window,  
 borderwidth=8,  
 relief='solid',  
 font='Times 22 bold',  
 background='red',  
 foreground='white',   
 text='Hola Mundo')  
  
label\_1.pack()

Se muestra de la siguiente forma. 

Estos atributos se han almacenado en un diccionario con sus respectivas key/value pairs:

|  |  |
| --- | --- |
| key | value |
| 'bd' | 8 |
| 'relief' | 'solid' |
| 'font' | 'Times 22 bold' |
| 'bg' | 'red' |
| 'fg' | 'white' |
| 'text' | 'Hola Mundo' |

Podemos acceder a los value a través de su key como en cualquier diccionario:

print(label\_1['bd'])

# Dynamically altering a Python tkinter label

Vamos a modificar las opciones de un Label después de que se haya empaquetado en la ventana principal. Incluso se pueden definir incluir opciones al diccionario que no estaban en la creación original del objeto.

label\_1 = Label(my\_window,  
 borderwidth=4,  
 relief='solid',  
 font='Times 22 bold',  
 background='red',  
 foreground='white',  
 text='Hola Mundo')  
  
label\_1.pack()  
  
# podemos acceder a los value a través de su key

print(label\_1['bd'])

# Modificamos las opciones, incluso de las que no existían  
label\_1['bg'] = 'blue'  
label\_1['text'] = 'Texto Cambiado'  
label\_1['padx'] = 20

Obteniendo algo así:



# The Python tkinter keys() method for a label

Mediante el método keys() sobre la instancia de label que hemos creado, obtenemos todos los keys asociados con la instancia de Label que nosotros llamamos label\_1.

print(label\_1.keys())

Obtenemos por consola:

['activebackground', 'activeforeground', 'anchor', 'background', 'bd', 'bg', 'bitmap', 'borderwidth', 'compound', 'cursor', 'disabledforeground', 'fg', 'font', 'foreground', 'height', 'highlightbackground', 'highlightcolor', 'highlightthickness', 'image', 'justify', 'padx', 'pady', 'relief', 'state', 'takefocus', 'text', 'textvariable', 'underline', 'width', 'wraplength']

Vemos que hay muchas opciones que no usaremos a la vez.

Alternativamente para ver **todas** las opciones de nuestra instancia podemos ejecutar el siguiente código:

**for** item **in** label\_1.keys():  
 print(**'{}: {}'**.format(item, label\_1[item]))

A continuación una muestra de lo obtenido por consola:

activebackground: SystemButtonFace

activeforeground: SystemButtonText

anchor: center

background: red

bd: 4

bg: red

bitmap:

borderwidth: 4

...

# Using StringVar and textvariable with a Python tkinter label

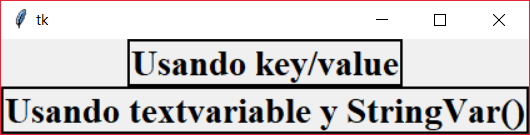
var\_1 = StringVar() *# crea un string que se puede asociar con un Label*label\_1 = Label(my\_window,  
 relief=**'solid'**,  
 font=**'Times 22 bold'**,  
 textvariable=var\_1)  
  
label\_1.pack()  
res = input(**'Mensaje: '**)  
var\_1.set(res)  
  
*# bucle principal*my\_window.mainloop()

Modificamos el valor de textvariable con el valor asignado a la variable res. Así podremos cambiar lo mostrado en el momento que queramos.

# The Python tkinter StringVar set() method

Hay dos formas de que nuestro código muestre el texto que queremos:

var\_1 = StringVar() *# crea un string que se puede asociar con un Label*label\_1 = Label(my\_window,  
 relief=**'solid'**,  
 font=**'Times 22 bold'**)  
label\_2 = Label(my\_window,  
 relief=**'solid'**,  
 font=**'Times 22 bold'**,  
 textvariable=var\_1)  
  
label\_1.pack()  
label\_2.pack()  
label\_1[**'text'**] = **'Usando key/value'**var\_1.set(**'Usando textvariable y StringVar()'**)

Obtenemos el mismo resultado pero es mejor aproximación usar StringVar() ya que usarnos variables.

# Introduction to the Python tkinter button widget

Un botón es un widget usado para la interacción con el usuario.

Por ejemplo si se pulsa el botón se ejecuta algún proceso. Se encuentra normalmente en una función (o método dentro de una clase) que define el proceso.

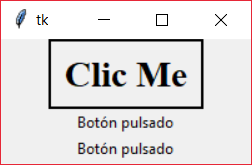
Mainloop está continuamente a la espera de eventos. Uno de dichos eventos (cierre de ventana, cambio de tamaño, etc. ) puede ser pulsar un botón.

Creamos el siguiente código, que despliega un botón.

**from** tkinter **import** \*  
  
**def** add\_label():  
 lable\_1 = Label(my\_window,  
 text=**'Botón pulsado'**)  
 lable\_1.pack()  
  
my\_window = Tk() *# crea un objeto por defecto*my\_window.geometry(**'200x100'**)  
  
button\_1 = Button(my\_window,  
 relief=**'solid'**,  
 font=**'Times 22 bold'**,  
 text=**'Clic Me'**,  
 **command=add\_label**) *# nombre de la función # sin paréntesis*button\_1.pack()  
  
*# bucle principal*my\_window.mainloop()

Obtenemos algo así en pantalla.



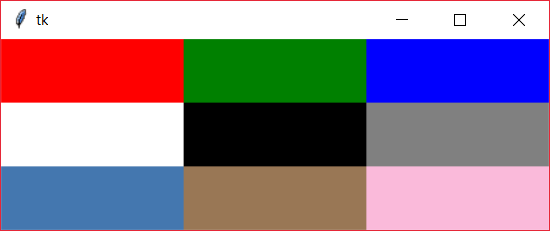
Cada vez que se pulsa el botón se ejecuta la función cuyo nombre aparece en command.

# The tkinter Grid Geometry Manager

Con el Grid Geometry Manager podemos dividir la ventana en una rejilla (regiones) ordenada por filas y columnas a nuestra elección.

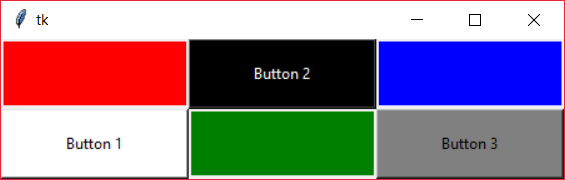
|  |  |  |
| --- | --- | --- |
| row=0  column=0 | row=0  column=1 | row=0  column=2 |
| row=1  column=0 | row=1  column=1 | row=1  column=2 |
| row=2  column=0 | row=2  column=1 | row=2  column=2 |

my\_window = Tk()  
  
label\_1 = Label(my\_window, width=**'20'**, height=**'3'**, bg=**'red'**)  
label\_2 = Label(my\_window, width=**'20'**, height=**'3'**, bg=**'green'**)  
label\_3 = Label(my\_window, width=**'20'**, height=**'3'**, bg=**'blue'**)  
label\_4 = Label(my\_window, width=**'20'**, height=**'3'**, bg=**'white'**)  
label\_5 = Label(my\_window, width=**'20'**, height=**'3'**, bg=**'black'**)  
label\_6 = Label(my\_window, width=**'20'**, height=**'3'**, bg=**'grey'**)  
label\_7 = Label(my\_window, width=**'20'**, height=**'3'**, bg=**'#4477af'**)  
label\_8 = Label(my\_window, width=**'20'**, height=**'3'**, bg=**'#997755'**)  
label\_9 = Label(my\_window, width=**'20'**, height=**'3'**, bg=**'#fabada'**)  
  
label\_1.grid(row=0, column=0)  
label\_2.grid(row=0, column=1)  
label\_3.grid(row=0, column=2)  
label\_4.grid(row=1, column=0)  
label\_5.grid(row=1, column=1)  
label\_6.grid(row=1, column=2)  
label\_7.grid(row=2, column=0)  
label\_8.grid(row=2, column=1)  
label\_9.grid(row=2, column=2)  
  
*# bucle principal*my\_window.mainloop()



También podemos disponer otros elementos aparte de Labels, como vemos en el siguiente ejemplo que alterna también botones.

my\_window = Tk()  
  
label\_1 = Label(my\_window, width=**'20'**, height=**'3'**, bg=**'red'**)  
label\_2 = Label(my\_window, width=**'20'**, height=**'3'**, bg=**'green'**)  
label\_3 = Label(my\_window, width=**'20'**, height=**'3'**, bg=**'blue'**)  
button\_1 = Button(my\_window, width=**'20'**, height=**'3'**, bg=**'white'**, text=**'Button 1'**)  
button\_2 = Button(my\_window, width=**'20'**, height=**'3'**, bg=**'black'**, text=**'Button 2'**, fg=**'white'**)  
button\_3 = Button(my\_window, width=**'20'**, height=**'3'**, bg=**'grey'**, text=**'Button 3'**)  
  
label\_1.grid(row=0, column=0)  
button\_2.grid(row=0, column=1)  
label\_3.grid(row=0, column=2)  
button\_1.grid(row=1, column=0)  
label\_2.grid(row=1, column=1)  
button\_3.grid(row=1, column=2)  
  
*# bucle principal*my\_window.mainloop()



# Designing a Simple tkinter GUI

<https://www.youtube.com/watch?v=XZ2G29ZUaII&index=27&list=PL6lxxT7IdTxGoHfouzEK-dFcwr_QClME_>

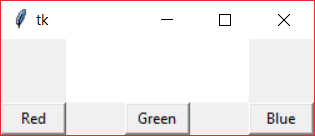
Ejercicio:

Produce a GUI using python and tkinter thas has three buttons and a label. Arrange for the clicking of a button to set the background colour of the label to red, green and blue

i.e. clicking one button changes the bakcground of the label to red, clicking the second button changes it to green and clicking the third button changes it to blue.

When the program first runs have the background of the label set at white. Have nothing displayed in the label and have the text of the buttons display Red, Gren an Blue respectively.

**from** tkinter **import** \*  
  
**def** set\_red():  
 label\_1[**'bg'**]=**'red'  
  
def** set\_green():  
 label\_1[**'bg'**]=**'green'  
  
def** set\_blue():  
 label\_1[**'bg'**]=**'blue'**my\_window = Tk()  
  
label\_1 = Label(my\_window, width=**'20'**, height=**'3'**, background=**'white'**)  
button\_red = Button(my\_window, text=**'Red'**, width=6, command=set\_red)  
button\_green = Button(my\_window, text=**'Green'**, width=6, command=set\_green)  
button\_blue = Button(my\_window, text=**'Blue'**, width=6, command=set\_blue)  
  
label\_1.grid(row=0, column=1)  
button\_red.grid(row=1, column=0)  
button\_green.grid(row=1, column=1)  
button\_blue.grid(row=1, column=2)  
  
my\_window.mainloop()

Obteniendo una ventana como la que sigue, que además al pulsar cada uno de los tres botones inferiores, cambia el color de blanco a rojo, verde o azul según el botón.

# The tkinter Entry Widget

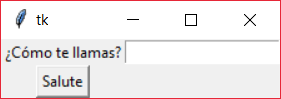
El widget **entry** se usa para introducir cadenas de texto a través de nuestra ventana. Para extraer el contenido de una entry se emplea el comando.

name = entry\_1.get()

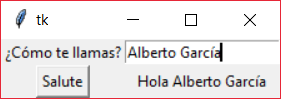
El programa quedaría como sigue. Nótese que no usamos stringvar() para este ejemplo aunque siempre sea recomendable, lo haremos más adelante con problemas más avanzados.

**from** tkinter **import** \*  
  
**def** salute():  
 name = entry\_1.get()  
 message = **'Hola {}'**.format(name)  
 label\_2[**'text'**] = message   
  
my\_window = Tk()  
  
label\_1 = Label(my\_window, text=**'¿Cómo te llamas?'**)  
entry\_1 = Entry(my\_window)  
button\_1 = Button(my\_window, text=**'Salute'**, command=salute)  
label\_2 = Label(my\_window)  
  
label\_1.grid(row=0, column=0)  
entry\_1.grid(row=0, column=1)  
button\_1.grid(row=1, column=0)  
label\_2.grid(row=1, column=1)  
  
my\_window.mainloop()

Se despliega al ejecutar, una ventana tal que así.



Y al escribir dentro de la sección definida para ello y pulsar el botón, obtenemos un cambio.



# The Entry widget and StringVar()

Esto es continuación del tema previo. Una mejora más elegante del mismo programa:

**from** tkinter **import** \*  
  
**def** salute():  
 *""" modify lable\_2's saludo """* var\_saludo.set(**'Hola '** + var\_name.get()) *# modificamos var\_1  
  
# variables*my\_window = Tk()  
var\_saludo = StringVar()  
var\_name = StringVar()  
  
*# build the GUI*label\_1 = Label(my\_window, text=**'¿Cómo te llamas?'**)  
entry\_1 = Entry(my\_window, textvariable=var\_name)  
button\_1 = Button(my\_window, text=**'Salute'**, command=salute)  
label\_2 = Label(my\_window, textvariable=var\_saludo)  
  
label\_1.grid(row=0, column=0)  
entry\_1.grid(row=0, column=1)  
button\_1.grid(row=1, column=0)  
label\_2.grid(row=1, column=1)  
  
my\_window.mainloop()

Obtenemos el mismo GUI y funciona igual.

## Pasos para construir GUI

1. Construir el GUI
2. Obtener los input del usuario desde el GUI para almacenarlos en variables del programa.
3. Procesar los datos que residen en dichas variables
4. Almacena los resultados del procesamiento de las variables.
5. Transfiere los resultados desde las variables hasta el GUI

Esta aproximación permite que el código de procesamiento esté separado del GUI. Permitiendo que el GUI sea alterado fácilmente sin afectar el código.

# Fahrenheit to Celsius Python GUI

Vamos a realizar un programa que mediante una GUI no convierta unidades de grados fahrenheit a celsius.

La fórmula se puede obtener de la siguiente fuente:

<https://es.wikipedia.org/wiki/Grado_Fahrenheit>

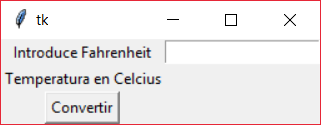
El ejemplo de este caso se mejora en el siguiente apartado y no merece la pena su desarrollo.

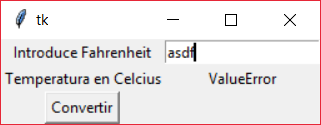
# Alternative Python Fahrenheit to Celsius GUI

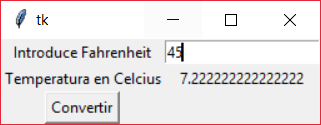
La fórmula se puede obtener de la siguiente fuente:

<https://es.wikipedia.org/wiki/Grado_Fahrenheit>

**from** tkinter **import** \*  
  
  
**def** fahrenheit\_to\_celcius(f):  
 **return** (f-32)/1.8  
  
  
**def** convert():  
 *""" convert fahrenheit to celsius """* **try**:  
 f=float(var\_fahenheit.get())  
 c = fahrenheit\_to\_celcius(f)  
 var\_celcius.set(c)  
 **except** ValueError:  
 var\_celcius.set(**'ValueError'**)  
 **else**:  
 print(**'else: cambio realizado'**)  
 **finally**:  
 print(**'finally: fin conversión'**)  
  
  
*# variables*my\_window = Tk()  
var\_fahenheit = StringVar()  
var\_celcius = StringVar()  
  
*# build the GUI*label\_fah = Label(my\_window, text=**'Introduce Fahrenheit'**)  
label\_cel = Label(my\_window, text=**'Temperatura en Celcius'**)  
button\_1 = Button(my\_window, text=**'Convertir'**, command=convert)  
entry\_1 = Entry(my\_window, textvariable=var\_fahenheit)  
label\_result = Label(my\_window, textvariable=var\_celcius)  
  
label\_fah.grid(row=0, column=0)  
label\_cel.grid(row=1, column=0)  
button\_1.grid(row=2, column=0)  
entry\_1.grid(row=0, column=1)  
label\_result.grid(row=1, column=1)  
  
my\_window.mainloop()







# Setting the focus within a Python GUI

Entendemos por focus que el widget de tipo Entry tenga el cursor parpadenado preparado para escribir en su posición.

Para pasar de un widget con el que se pueda interactuar a otro pulsamos en el teclado el botón TAB, una vez estamos en el que queremos podemos activar con SPACE un botón. Antes teníamos que emplear el ratón para seleccionar el entry en cuestión.

Para que la ventana empiece con el focus donde lo deseamos basta con incluir la siguiente orden antes de mainloop.

**entry\_1.focus()** # este elemento entry\_1 tendrá el cursor sobre él  
my\_window.mainloop()

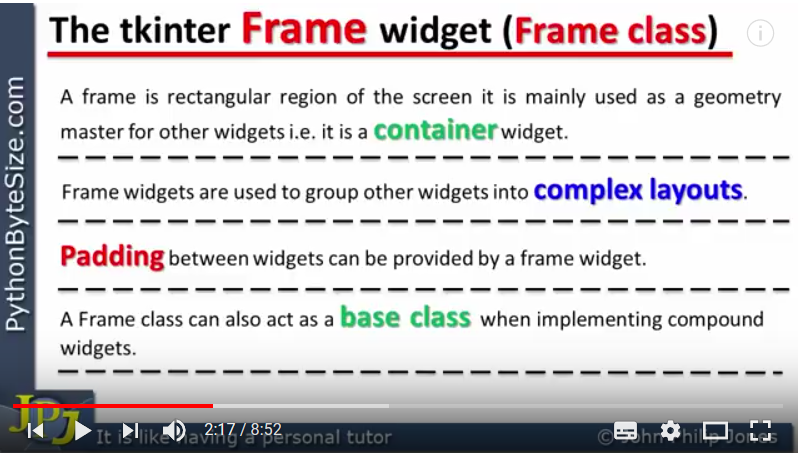
# Focus and Tab order of Widgets in a Python GUI

Vamos a configurar como con la orden TAB como pasamos de un focus a otro.

**En función de el orden en que se creaen las diferentes instancias de Entry en nuestro código, será como irá desplazando el focus a medida que pulsamos TAB**.

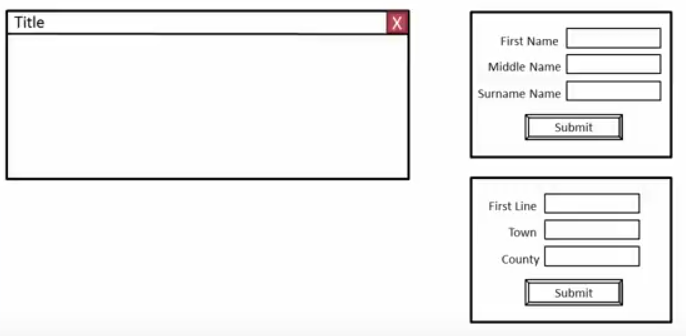
# Introduction to the Python tkinter Frame widget

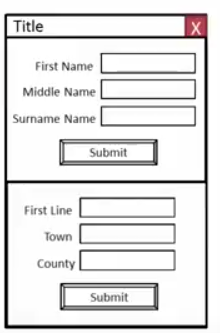
https://youtu.be/HyKaZggBAU4?list=PL6lxxT7IdTxGoHfouzEK-dFcwr\_QClME\_



Usando un Frame podemos englobar parte de la lógica de nuestro programa incluyendo los Label, Entry, Buttons, etc. que consideremos necesario. Para luego incluir nuestro Frame en la ventana principal junto con otros widgets.

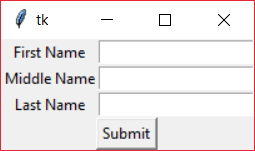
Dentro de nuestro Frame los row/columns de sus elementos son relativos al Frame, no a la ventana principal. El método grid se emplea sobre los Frame y no sobre los widgets individuales que lo componen.





# Building a Python tkinter GUI using Frames

**from** tkinter **import** \*  
  
my\_window = Tk()  
  
frame\_name = Frame(my\_window)  
  
*# build the GUI  
# Name Frame*label\_first = Label(frame\_name, text=**'First Name'**)  
label\_middle = Label(frame\_name, text=**'Middle Name'**)  
label\_surname = Label(frame\_name, text=**'Last Name'**)  
  
entry\_first = Entry(frame\_name)  
entry\_middle = Entry(frame\_name)  
entry\_surname = Entry(frame\_name)  
  
button\_submit\_name = Button(frame\_name, text=**'Submit'**)  
  
label\_first.grid(row=0, column=0) *# respecto a su frame\_name*label\_middle.grid(row=1, column=0)  
label\_surname.grid(row=2, column=0)  
  
entry\_first.grid(row=0, column=1)  
entry\_middle.grid(row=1, column=1)  
entry\_surname.grid(row=2, column=1)  
  
button\_submit\_name.grid(row=3, columnspan=2)  
  
frame\_name.grid(row=0, column=0) *# respecto a my\_window  
  
# entry\_first.focus()*my\_window.mainloop()

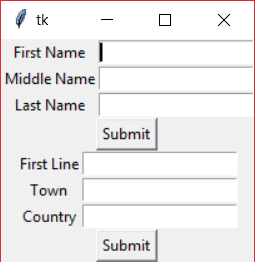


# Adding Frames to a Python tkinter window

Continuamos el ejemplo anterior para ahora completar la ventana incluyento 2 Frames que podemos colocar donde queramos. En este caso en vertical.

**from** tkinter **import** \*  
  
*# main window and 2 frames*my\_window = Tk()  
frame\_name = Frame(my\_window)  
frame\_address = Frame(my\_window)  
  
*# build the GUI  
# Name Frame*label\_first = Label(frame\_name, text=**'First Name'**)  
label\_middle = Label(frame\_name, text=**'Middle Name'**)  
label\_surname = Label(frame\_name, text=**'Last Name'**)  
  
entry\_first = Entry(frame\_name)  
entry\_middle = Entry(frame\_name)  
entry\_surname = Entry(frame\_name)  
  
button\_submit\_name = Button(frame\_name, text=**'Submit'**)  
  
label\_first.grid(row=0, column=0) *# respecto a su frame\_name*label\_middle.grid(row=1, column=0)  
label\_surname.grid(row=2, column=0)  
  
entry\_first.grid(row=0, column=1)  
entry\_middle.grid(row=1, column=1)  
entry\_surname.grid(row=2, column=1)  
  
button\_submit\_name.grid(row=3, columnspan=2)  
  
*# address frame*label\_first\_line = Label(frame\_address, text=**'First Line'**)  
label\_town = Label(frame\_address, text=**'Town'**)  
label\_country = Label(frame\_address, text=**'Country'**)  
  
entry\_first\_line = Entry(frame\_address)  
entry\_town = Entry(frame\_address)  
entry\_country = Entry(frame\_address)  
  
button\_submit\_address = Button(frame\_address, text=**'Submit'**)  
  
label\_first\_line.grid(row=0, column=0) *# respecto a su frame\_name*label\_town.grid(row=1, column=0)  
label\_country.grid(row=2, column=0)  
  
entry\_first\_line.grid(row=0, column=1)  
entry\_town.grid(row=1, column=1)  
entry\_country.grid(row=2, column=1)  
  
button\_submit\_address.grid(row=3, columnspan=2)  
  
*# adding frames to main window*frame\_name.grid(row=0, column=0) *# respecto a my\_window*frame\_address.grid(row=1, column=0)  
  
*# Focus*entry\_first.focus()  
  
my\_window.mainloop()

Obteniendo.



# Creating and positioning a widget with one line of Python code

Para ahorrar lineas de código y así hacerlo más compacto y legible se pueden unificar estas dos lineas:

# create Label

label\_first = Label(frame\_name, text=**'First Name'**)

# positioning Label

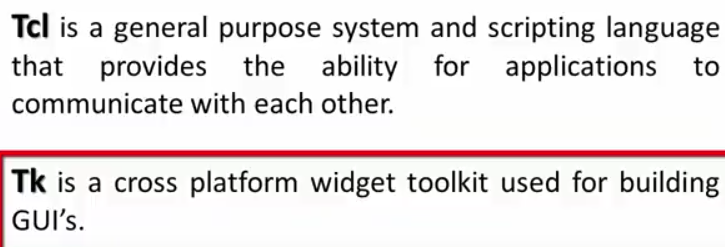
label\_first.grid(row=0, column=0)

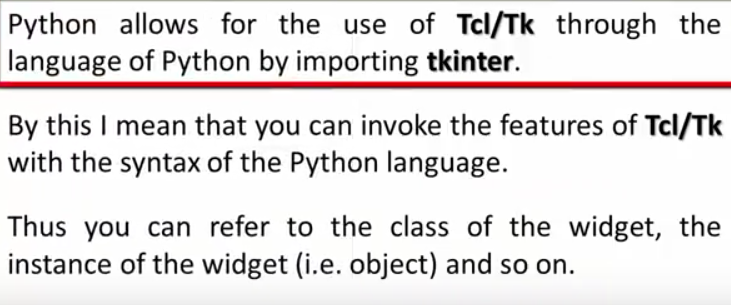
Mediante el siguiente código podemos crear y posicionar Label en una solo línea. No siempre es la mejor de las recomendaciones.

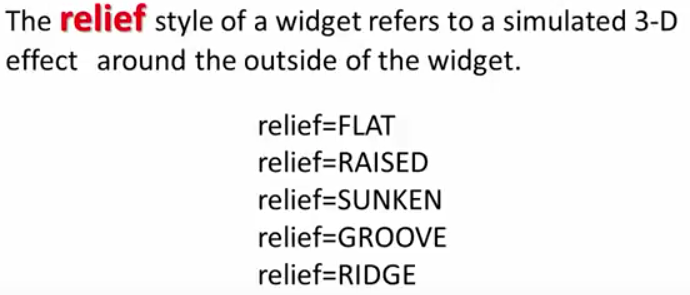
label\_first = Label(frame\_name, text=**'First Name'**).grid(row=0, column=0)

Esto puede usarse para todos los widgets que usan grid.

# Creating instances of tkinter widgets using Object Oriented Python







# Creating instances of the tkinter Frame using OOP (object Oriented Paradigm)

https://youtu.be/8loDDEJMCdw?list=PL6lxxT7IdTxGoHfouzEK-dFcwr\_QClME\_

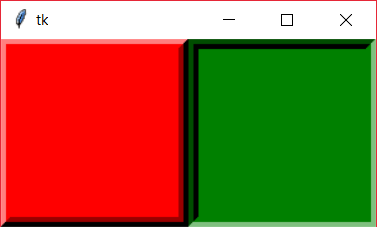
# How to Inherit from a Python tkinter Frame

Observando el siguiente código:

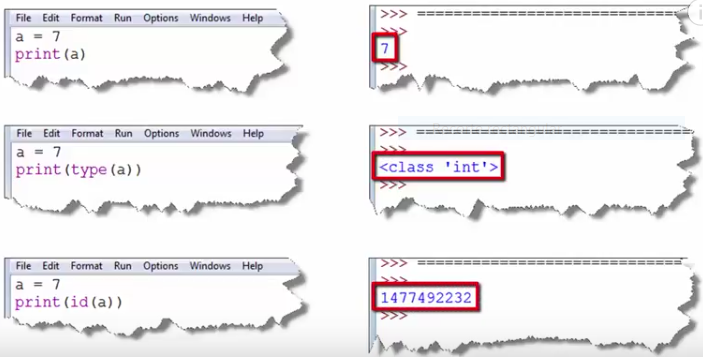
**from** tkinter **import** \*  
  
# main window  
my\_window = Tk()  
  
# build the GUI  
frame\_a = Frame(my\_window, height=150, width=150, relief=RAISED, bd=8, bg='red')  
frame\_b = Frame(my\_window, height=150, width=150, relief=SUNKEN, bd=8, bg='green')  
  
frame\_a.grid(row=0, column=0)  
frame\_b.grid(row=0, column=1)  
  
my\_window.mainloop()

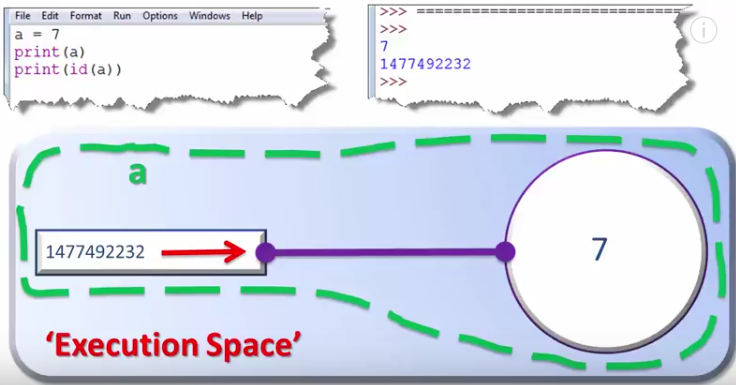
Se puede apreciar que ambos Frames: frame\_a y frame\_b tienen propiedades comunes (height, width, bd) y propiedades diferentes (reliev, bg). Podemos crear una clase que herede de Frame con las opcoines comunes y ajustar las propiedades diferentes a gusto.

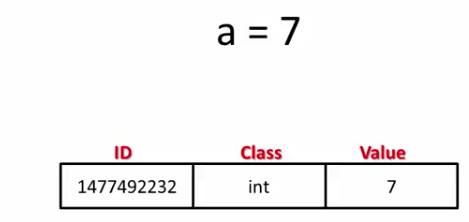
**from** tkinter **import** \*  
  
**class** TipicalFrame(Frame):  
 **def** \_\_init\_\_(self, the\_window, \*\*kwargs):  
 super().\_\_init\_\_(\*\*kwargs)  
 self[**'height'**]=150  
 self[**'width'**]=150  
 self[**'bd'**]=8  
  
  
*# main window*my\_window = Tk()  
  
*# build the GUI*frame\_a = TipicalFrame(my\_window, relief=RAISED, bg=**'red'**)  
frame\_b = TipicalFrame(my\_window, relief=SUNKEN, bg=**'green'**)  
  
frame\_a.grid(row=0, column=0)  
frame\_b.grid(row=0, column=1)  
  
my\_window.mainloop()



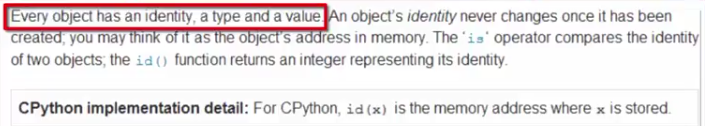
# The id and type of a Widget that inherits from a Python tkinter Frame

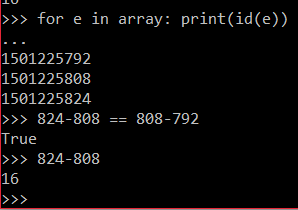
Repasando





'a' vale 7, su id apunta a donde el objeto está en memoria. 'int' es el tipo de clase (en python los tipos están basados en clases)





Id puede considerarse (y en Cpython así es) como la dirección de memoria donde el valor está almacenada.

Podemos comprobarlo con el siguiente código:

**from** tkinter **import** \*  
  
**class** TipicalFrame(Frame):  
 **def** \_\_init\_\_(self, the\_window, \*\*kwargs):  
 super().\_\_init\_\_(\*\*kwargs)  
 self[**'height'**]=150  
 self[**'width'**]=150  
 self[**'bd'**]=8  
  
  
*# main window*my\_window = Tk()  
  
*# build the GUI*frame\_a = TipicalFrame(my\_window, relief=RAISED, bg=**'red'**)  
frame\_b = TipicalFrame(my\_window, relief=SUNKEN, bg=**'green'**)  
  
frame\_a.grid(row=0, column=0)  
frame\_b.grid(row=0, column=1)  
  
*# id and type*print(**'id of frame\_a is {}'**.format(id(frame\_a)))  
print(**'type of frame\_a is {}'**.format(type(frame\_a)))  
  
my\_window.mainloop()

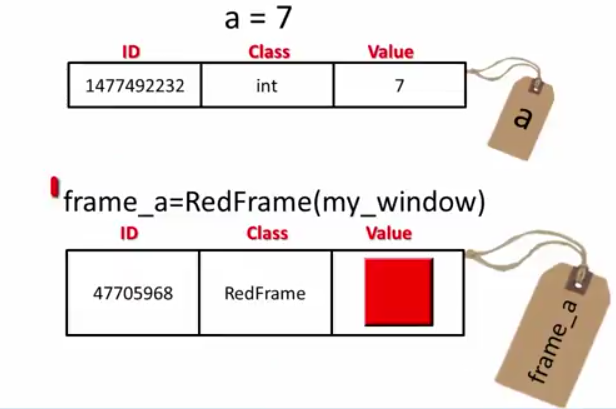
Por consolo nos muestra lo siguiente:

>>>

id of frame\_a is 59746384

type of frame\_a is <class '\_\_main\_\_.TipicalFrame'>

Esto se puede explicar por similitud:



# The need for the \_\_init\_\_ method and self parameter when inheriting with tkinter

https://youtu.be/2a-WeQMZMLg?list=PL6lxxT7IdTxGoHfouzEK-dFcwr\_QClME\_

El hecho de crear un objeto tipo RedFrame invoca \_\_init\_\_ automáticamente, no hay que llamarlo.

'self' recibe el id del objeto creado con frame\_b = Frame(my\_window). 'self' dentro de \_\_init\_\_ se refiere a la instancia que se está creando.

**from** tkinter **import** \*  
  
**class** TipicalFrame(Frame):  
 **def** \_\_init\_\_(self, the\_window, \*\*kwargs):  
 super().\_\_init\_\_(\*\*kwargs)  
 self[**'height'**]=150  
 self[**'width'**]=150  
 self[**'bd'**]=8  
 print(**'id of self is {}'**.format(id(self)))  
  
  
*# main window*my\_window = Tk()  
  
*# build the GUI*frame\_a = TipicalFrame(my\_window, relief=RAISED, bg=**'red'**)  
print(**'id of frame\_a is {}'**.format(id(frame\_a)))  
  
frame\_b = TipicalFrame(my\_window, relief=SUNKEN, bg=**'green'**)  
print(**'id of frame\_b is {}'**.format(id(frame\_b)))  
  
  
frame\_a.grid(row=0, column=0)  
frame\_b.grid(row=0, column=1)  
  
my\_window.mainloop()

Por consola observamos:

>>>

id of self is 60074032

id of frame\_a is 60074032

id of self is 60073648

id of frame\_b is 60073648

# Using the Python help function to assist with inheritance

Vamos a usar la función help() de python para ver que clases internas tenemos en la estructura interna de clases de tkinter.

**from** tkinter **import** \*  
  
print(help(Frame))

Obtenemos por consola, entre otras cosas:

Help on class Frame in module tkinter:

class Frame(Widget)

| Frame widget which may contain other widgets and can have a 3D border.

|

| Method resolution order:

| Frame

| Widget

| BaseWidget

| Misc

| Pack

| Place

| Grid

| builtins.object

|

| Methods defined here:

|

Vemos los parámetros que se pueden pasar al inicializar instancia o para usar herencia.

| \_\_init\_\_(self, master=None, cnf={}, \*\*kw)

| Construct a frame widget with the parent MASTER.

|

Y lo siguiente se herede la clase padre y pueden usarse en el hijo.

| Valid resource names: background, bd, bg, borderwidth, class,

| colormap, container, cursor, height, highlightbackground,

| highlightcolor, highlightthickness, relief, takefocus, visual, width.

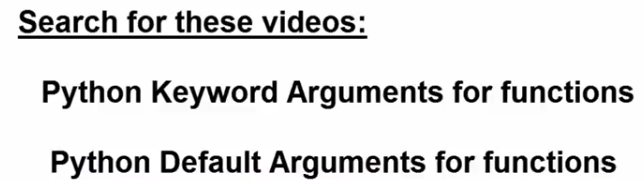
Una explicación más detallada:

| \_\_init\_\_(self, master=None, cnf={}, \*\*kw)

Podemos ver 4 parámetros:

* self
* master=None
* cnf={} permite pasar un diccionario de key/value de entre los Valid resource names.
* \*\*kw permite pasar una cantidad variable de keyword como argumento

Por ahora solo nos centramos en self y en master. Para más información sobre los otros dos:



Herencia múltiple. Si seguimos leyendo el resultado de help(Frame) vemos que hereda de dos clases.

| ----------------------------------------------------------------------

| Methods inherited from **BaseWidget**:

|

| destroy(self)

| Destroy this and all descendants widgets.

|

| ----------------------------------------------------------------------

| Methods inherited from **Misc**:

|

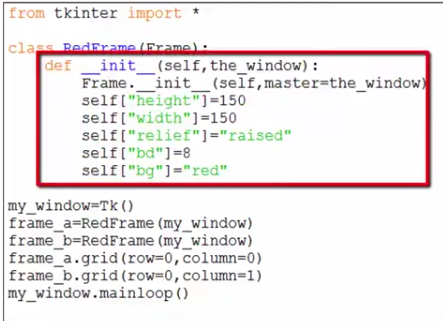
| \_\_getitem\_\_ = cget(self, key)

| Return the resource value for a KEY given as string.

|

| \_\_repr\_\_(self)

| Return repr(self).

Veamos código. Obervemos la inicialización, que se ve self, master a Frame.\_\_init

print(help(Frame))

print(help(RedFrame))

proporcionarán similares resultados, pero este último aportará el \_\_init\_\_ que ha definido en su clase.

Print(help()) puede ayudarnos mucho para entender la estructura con la que estamos trabajando.

# Using Inheritance to Build a Python tkinter GUI

**from** tkinter **import** \*  
  
**class** HelloNameFrame(Frame):  
 **def** \_\_init\_\_(self, the\_window):  
 Frame.\_\_init\_\_(self, the\_window)  
 self.users\_name = StringVar()  
 self.display\_string = StringVar()  
  
 self.friendly\_label = Label(self, text=**'Enter your name: '**)  
 self.name\_entry = Entry(self, textvariable=self.users\_name)  
 self.button = Button(self, text=**'Click Me'**, command=self.display\_output)  
 self.display\_label = Label(self, textvariable=self.display\_string, relief=**'solid'**)  
  
 self.friendly\_label.grid(row=0, column=0)  
 self.name\_entry.grid(row=0, column=1)  
 self.button.grid(row=1, column=0)  
 self.display\_label.grid(row=1, column=1)  
  
 **def** display\_output(self):  
 self.display\_string.set(**'Hola '** + self.users\_name.get())  
  
my\_window = Tk()  
frameA = HelloNameFrame(my\_window)  
frameA.grid(row=0, column=0)  
  
my\_window.mainloop()

El código siguiente nos dice que HelloNameFrame está usando la rutina de inicialización de la superclase Frame.

Frame.\_\_init\_\_(self, the\_window)

Esto es recomendable emplear herencia por reutilización de código y mayor claridad. Se puede considerar que **HelloNameFrame** es una factoría de estas instancias.

Se pasa a display\_output como parámetro self, para que cada instancia sepa el identificador sobre el que debe trabajar.