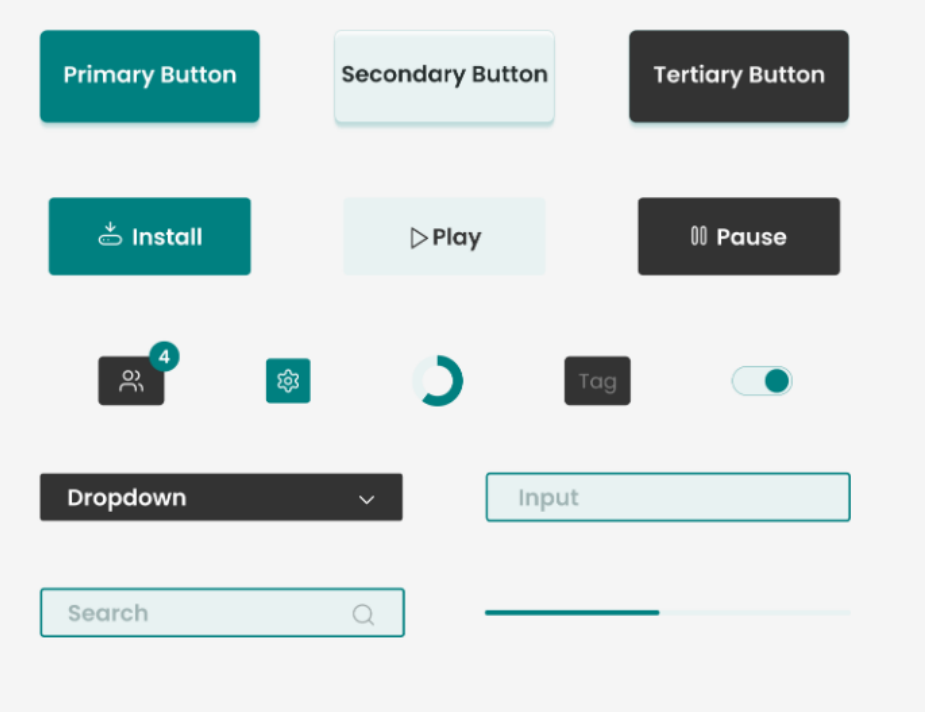
PYTHON

GUI Programming

Tkinter Essentials

TKINTER ESSENTIALS

- ¿Qué significa GUI? → Graphical User Interface



TKINTER ESSENTIALS

- Algo de historia de las interfaces de usuario: TERMINALES

IMB 3270

VT100

TKINTER ESSENTIALS

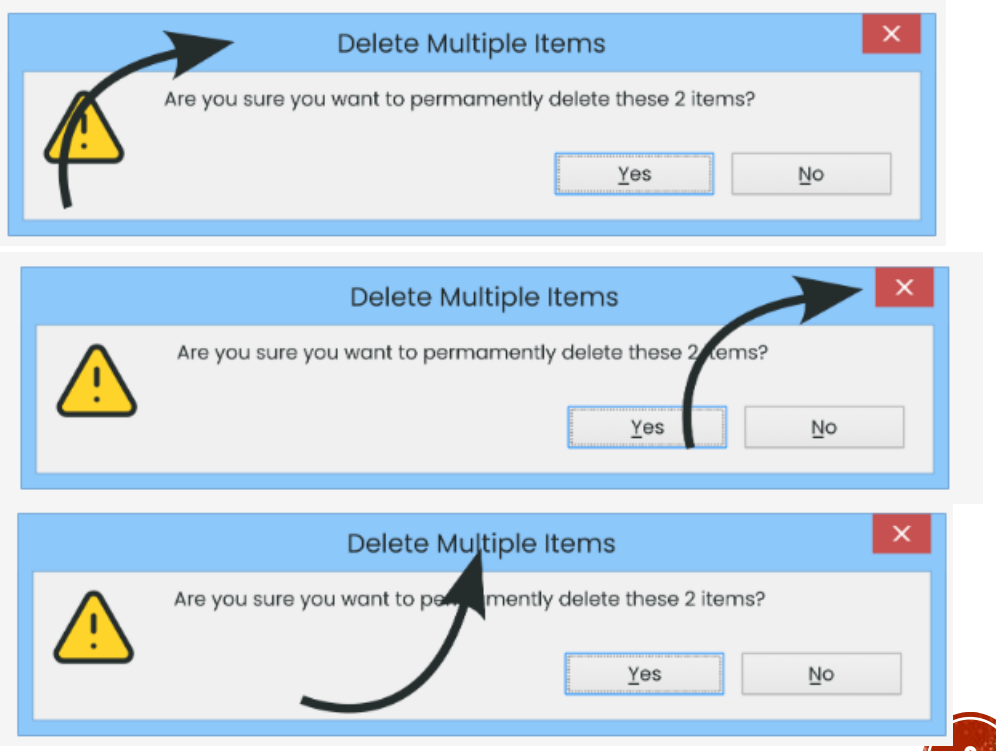
- Algo de historia: la terminal
  - Monocromática: fondo negro y caracteres de un color.
  - Únicamente letras, dígitos y algún otro carácter sencillo.
  - No disponen de ratones, *trackballs* o pantallas táctiles.
  - Algunos disponen de lápiz óptico.
  - Típicamente 80x25.
  - Demasiadas limitaciones. Solución → GUI

TKINTER ESSENTIALS

- Programación visual:
  - Aplicaciones capaces de utilizar características de GUI.
  - Ofrecen una o más ventanas visibles en pantalla.
  - En algunos entornos limitación a una única ventana a pantalla completa: dispositivos móviles.
  - Suelen contener datos comunes:
    - Barras de títulos, marcos, botones, iconos, etc.

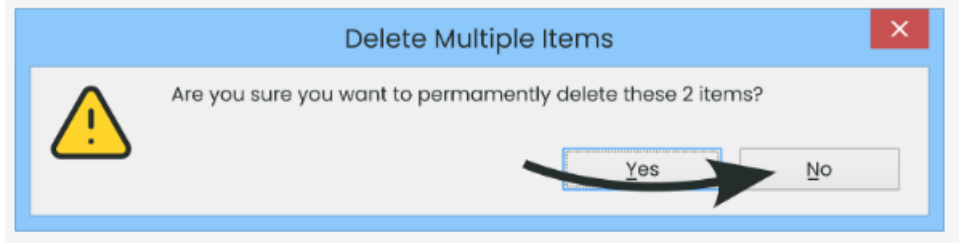
TKINTER ESSENTIALS

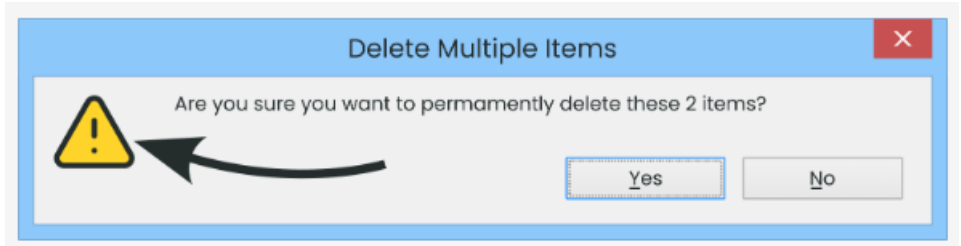
- Controles (Widgets):
  - *Focused widget*. Es el control que posee el foco, destinatario de todas o algunas de las acciones del usuario.
  - Componentes típicos de las ventanas:
    - Barra de título
    - Botón de cerrar
    - Título de la ventana



TKINTER ESSENTIALS

- Controles (Widgets):
  - El interior de la ventana está compuesto por *widgets*, que pueden ser **activos** (se puede interactuar con ellos, como los botones) como **no activos** (solo son informativos, como los iconos).



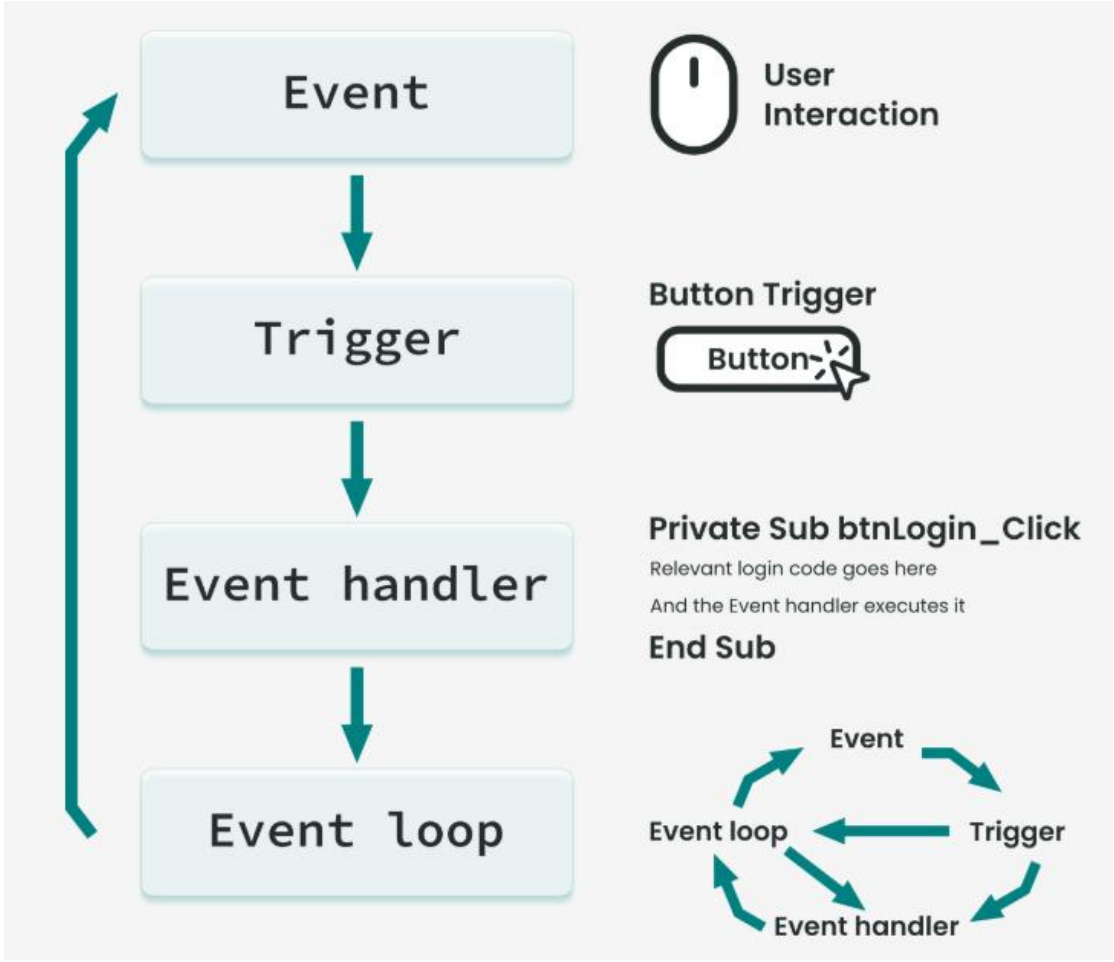


TKINTER ESSENTIALS

- **EDP**: ***Event-driven paradigm***
  - Detectar, registrar y clasificar todas las acciones del usuario no es responsabilidad del programador → el controlar de eventos (*event controller*) es el encargado.
  - El programador debe comunicar al controlador de eventos qué quiere que ocurra cuando ocurra un evento concreto a través de los manejadores de eventos (*evento handler*).
  - Los manejadores de eventos solo se programan para los eventos que se quieren gestionar.

TKINTER ESSENTIALS

- **Eventos**
  - Algunos ejemplos:
    - Pulsar el botón izquierdo del ratón
    - Mover el ratón
    - Pulsar una tecla
    - Tocar una pantalla
    - Cambiar el foco
    - …



TKINTER ESSENTIALS

- **Kit de desarrollo de GUI**
  - Cada sistema operativo ofrece su propio *kit* o *framework* nativo de servicios GUI.
  - También existen *frameworks* multiplataforma, como **GTK** (*Gimp Toolkit*) o **Qt**, utilizados en los entornos Unix/Linux.
  - El desarrollo de aplicaciones multiplataforma con GUI, requiere adaptadores. A este tipo de adaptador se le denomina kit de herramientas de widgets, kit de herramientas de GUI o biblioteca de UX.
  - Uno de estos conjuntos de herramientas es **Tk** (*ToolKit*).

TKINTER ESSENTIALS

- **Tk y TkInter. Características.**
  - Gratuito y abierto.
  - Maduro (disponible desde 1991)
  - Dispone de más de 30 *widgets*.
  - Disponible en muchos lenguajes, incluido Python.
  - TkInter → *Wrapper* de Tk para Python.
  - TkInter → Tk Interface.
  - TkInter → Forma parte de la biblioteca estándar de Python.
  - Idle está escrito con TkInter.

TKINTER ESSENTIALS

- **TkInter.**
  - Una aplicación con GUI TkInter contiene cuatro pasos esenciales:
    - Importar los compontes tkinter.
    - Crear la Ventana principal de la aplicación.
    - Añadir los *widgets*.
    - Lanzar el controlador de eventos.

TKINTER ESSENTIALS

- **TkInter.**
  - Importar el módulo:
    - import tkinter
    - import tkinter as tk
    - from tkinter import Button
    - from tkinter import * (arriesgado)
  - No importa todo. Hay que hacer importaciones explícitas de **módulos dentro del módulo tkinter**. Ejemplos:
    - ttk (*Tk themed widgets*), messagebox, filedialog, colorchooser, simpledialog, scrolledtext, dnd (drag and drop)

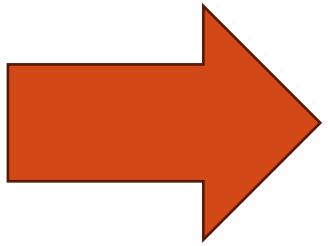
TKINTER ESSENTIALS

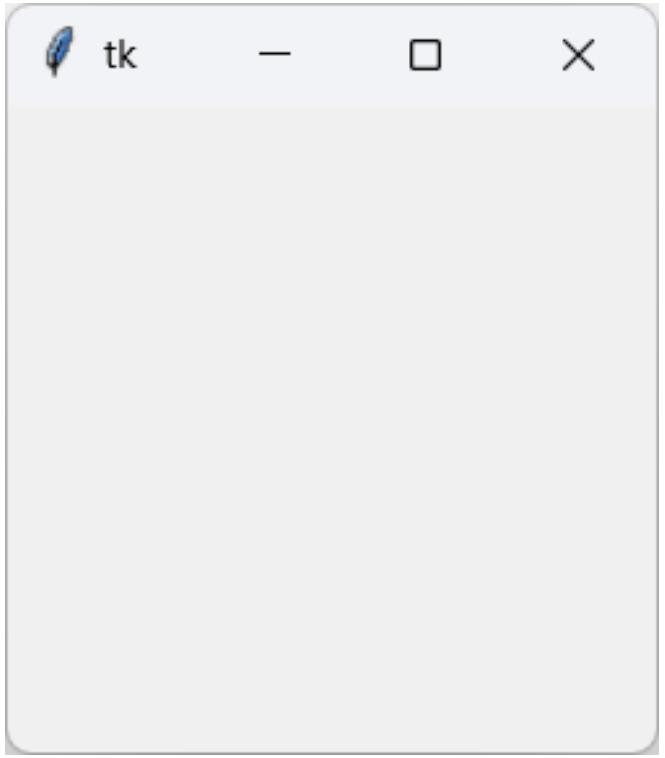
- **TkInter.**
  - **Creación de la ventana principal.** Normalmente es la única ventana. Se construye a partir del método **Tk().**
  - **Ejecución del bucle principal de la aplicación.**
    - Se ejecuta sobre la ventana principal mediante la invocación del método **mainloop()**.
    - El programa entra en un bucle infinito.
    - El programa queda a la espera eventos del usuario
    - Ejecuta los controladores de eventos (funciones asociadas a botones, teclas, etc.).
    - Redibuja la interfaz gráfica cuando es necesario.

TKINTER ESSENTIALS

- **TkInter.**

```
import tkinter as tk

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





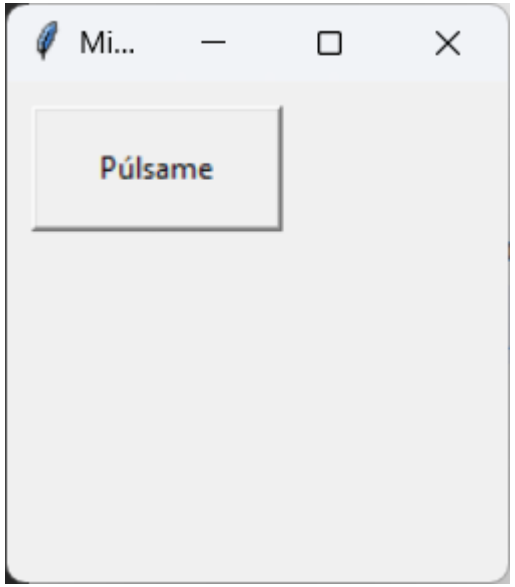
TKINTER ESSENTIALS

- **TkInter.**
  - Métodos de Tk (ventana):
    - title → main_window.title('Título de la ventana)
  - Botón:
    - Creación con **tk.Button***(main_window)*
    - Obligatoriamente se debe indicar la ventana a la que pertenece.
    - La creación no implica presentación → método **place** o **pack**.

TKINTER ESSENTIALS

TkInter.

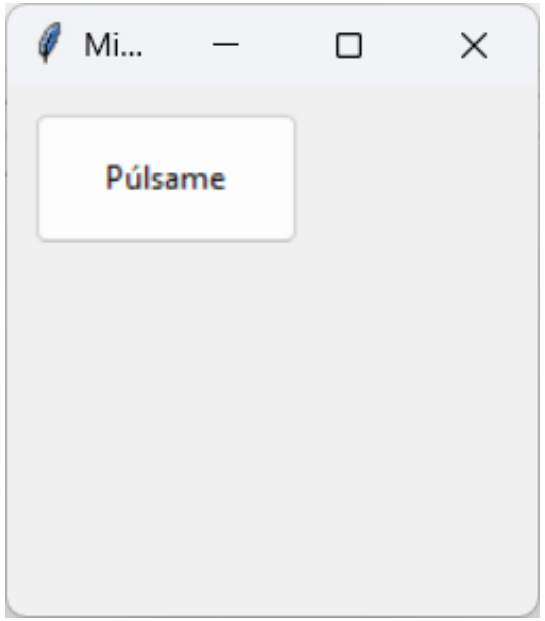
```
import tkinter as tk

main_window = tk.Tk()
main_window.title("Mi primera aplicación")

first_button = tk.Button(main_window, text='Púlsame')
first_button.place(x=10, y=10, width=100, height=50)

main_window.mainloop()
```



TKINTER ESSENTIALS

TkInter. Uso de ttk.

```
import tkinter as tk
from tkinter import ttk

main_window = tk.Tk()
main_window.title("Mi primera aplicación")

first_button = ttk.Button(main_window, text='Púlsame')
first_button.place(x=10, y=10, width=100, height=50)

main_window.mainloop()
```



TKINTER ESSENTIALS

- **TkInter. Sistema de coordenadas.**
  - La posición de referencia del widget es la esquina superior izquierda.
  - El tamaño del widget se asigna según su contenido (salvo que se especifique explícitamente).
  - Las coordenadas se indican en píxeles.
  - El origen se encuentra en la parte superior izquierda de la ventana, no en la inferior izquierda de un sistema cartesiano tradicional de dos dimensiones.

TKINTER ESSENTIALS

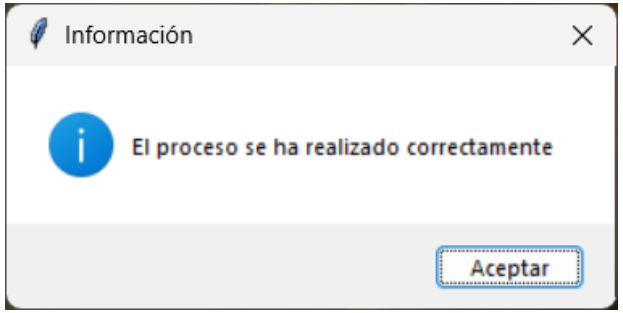
- **TkInter. *Event handler*.**
  - El controlador de eventos se puede implementar en una función (incluídas las lambdas) → función *callback*.
  - El atributo **command** permite asignar la función *callback* asociada al *widget* (lo admiten *Button*, C*heckbutton*, *Radiobutton, Scale (*deslizador*), Spinbox (*selector numérico*) y Menu)*.
  - Hay otros métodos de asignar funciones *callback* además del atributo **command**.

TKINTER ESSENTIALS

- **TkInter. *Event handler*.**

```
import tkinter as tk

def salir():
    main_window.destroy()

main_window = tk.Tk()
main_window.title("Mi primera aplicación")

first_button = tk.Button(main_window, text='Púlsame', command=salir)
first_button.place(x=10, y=10)

main_window.mainloop()
```

TKINTER ESSENTIALS

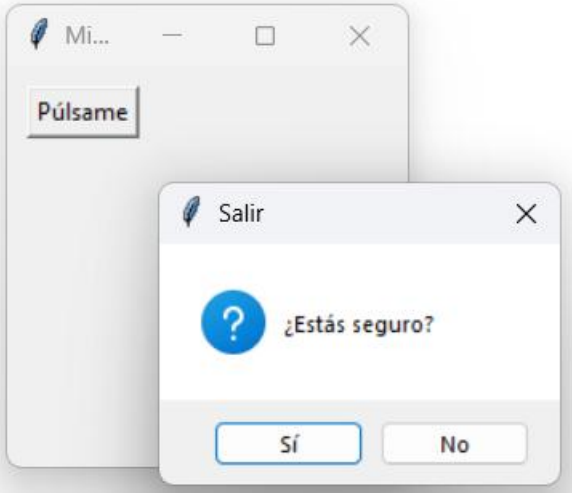


- **TkInter. Módulo *messagebox*.**
  - Permite crear cuadros de diálogo para mostrar mensajes, hacer preguntas y recibir respuestas del usuario.
  - Requiere una importación explícita (es un módulo).
  - El cuadro de diálogo es modal.
  - Función **showinfo** → Muestra un diálogo informativo con un botón aceptar.
  - Función **askquestion** → Devuelve **'yes'** si pulsa el botón con el texto afirmativo (despende del idioma).
    - NOTA: La constante tkinter.YES es 1

TKINTER ESSENTIALS

- **TkInter. Módulo *messagebox*.**

```
import tkinter as tk
from tkinter import messagebox

def salir():
    salir = messagebox.askquestion("Salir", "¿Estás seguro?")
    print(salir)
    if salir == 'yes':
        main_window.destroy();

main_window = tk.Tk()
main_window.title("Mi primera aplicación")

first_button = tk.Button(main_window, text='Púlsame', command=salir)
first_button.place(x=10, y=10)

main_window.mainloop()
```

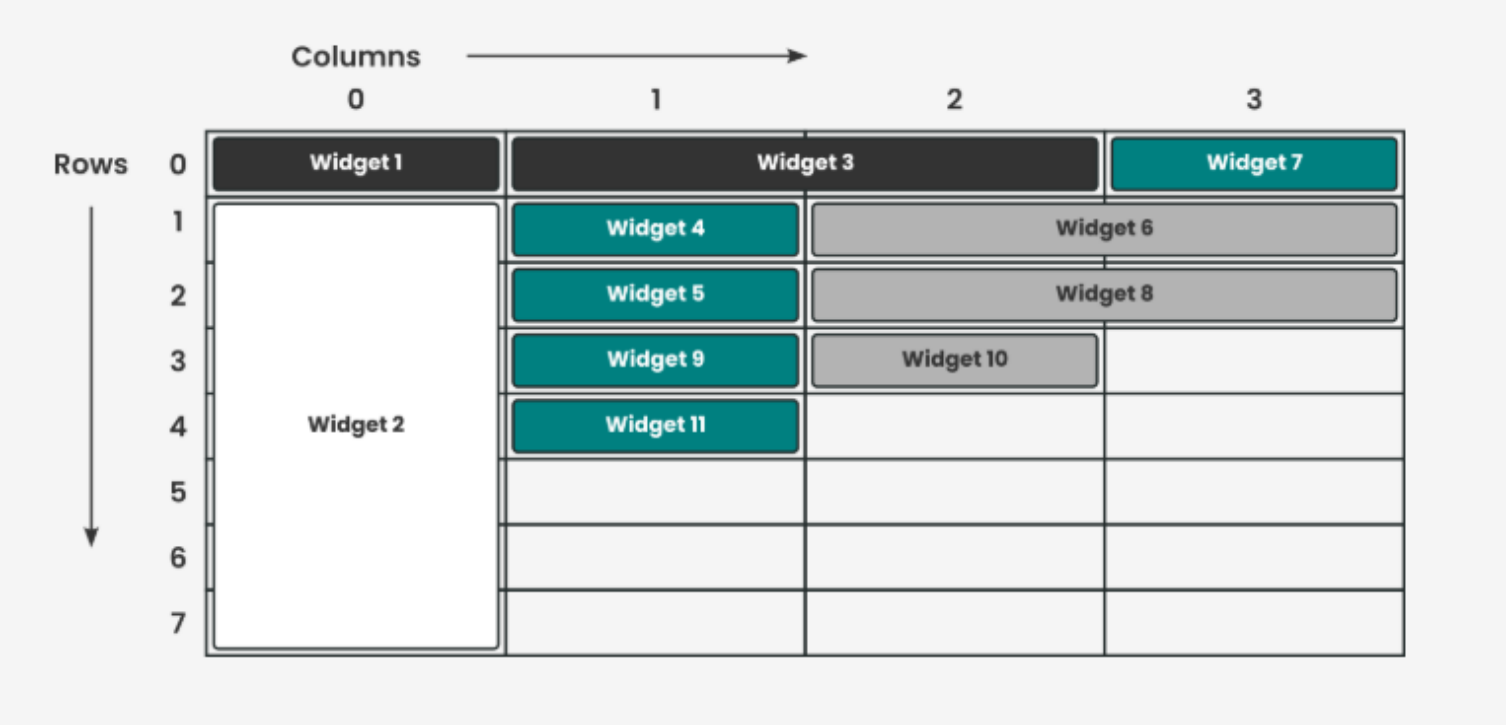


TKINTER ESSENTIALS

- **Gestores de geometrías (Geometry Managers)**
  - Son los mecanismos de TkInter para ubicar los widgets en la interfaz:
    - Gestor de geometría basado en **place**→Absolutamente manual. Preciso, requiere indicar la posición a nivel de píxeles. Riesgo de solapamientos y de ubicaciones fuera de la ventana.
    - Gestor de geometría **pack**→Automatiza parcialmente la distribución de los widgets. Aunque sus reglas son fijas, puede no tener el comportamiento deseado.
    - Gestor de geometría **grid**→ No es tan preciso como **place** pero es más flexible que **pack**.
    - NOTA: No se deben mezclar.

TKINTER ESSENTIALS

- **Gestores de geometrías (Geometry Managers)**
  - Ejemplo de distribución basada en el gestor de geometrías **grid.**



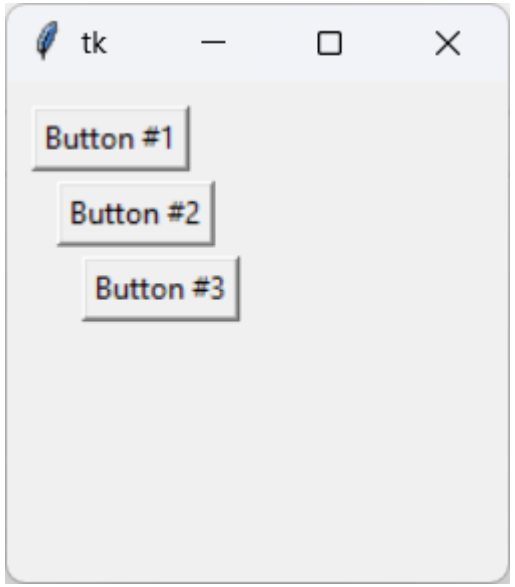
TKINTER ESSENTIALS

- **Gestores de geometrías (Geometry Managers)**
  - **Place**
  - Este gestor de geometría se basa en el uso del método **place**. Este método se invoca en los propios widgets, no en la ventana. El widget sabe a qué ventana pertenece ya que se le indica en el constructor como primer argumento.
  - **Principales parámetros** (pasados siempre como argumentos *keyword*)
    - **height**=h – Altura en píxeles. Si no se indica, se asigna automaticamente.
    - **width**=w – Altura en píxeles. Si no se indica, se asigna automaticamente.
    - **x**=x – Posición horizontal del píxel superior izquierdo relativo a la izquierda de la ventana.
    - **y**=y – Posición vertical del píxel superior izquierdo relativo a la parte superior de la ventana.

TKINTER ESSENTIALS

- **Gestores de geometrías (Geometry Managers)**
  - **Place. Ejemplo:**

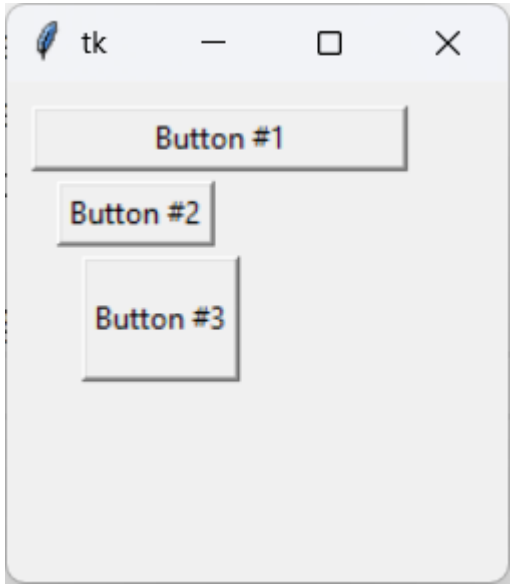
```
import tkinter as tk

window = tk.Tk()
button_1 = tk.Button(window, text="Button #1")
button_2 = tk.Button(window, text="Button #2")
button_3 = tk.Button(window, text="Button #3")
button_1.place(x=10, y=10)
button_2.place(x=20, y=40)
button_3.place(x=30, y=70)
window.mainloop()
```



TKINTER ESSENTIALS

- **Gestores de geometrías (Geometry Managers)**
  - **Place. Ejemplo:**

```
import tkinter as tk

window = tk.Tk()
button_1 = tk.Button(window, text="Button #1")
button_2 = tk.Button(window, text="Button #2")
button_3 = tk.Button(window, text="Button #3")
button_1.place(x=10, y=10, width=150)
button_2.place(x=20, y=40)
button_3.place(x=30, y=70, height=50)
window.mainloop()
```



TKINTER ESSENTIALS

- **Gestores de geometrías (Geometry Managers)**
  - **Grid**
  - Comparte la responsabilidad con el desarrollador.
  - **Grid** distribuye el contenedor en una cuadrícula:
    - La cuadrícula no es visible.
    - Las filas tienen el mismo alto.
    - Las columnas tienen el mismo ancho.
    - No se indican el número de filas y columnas de la cuadrícula. La dimensión de la cuadrícula viene determinada por la posición de los widgets.

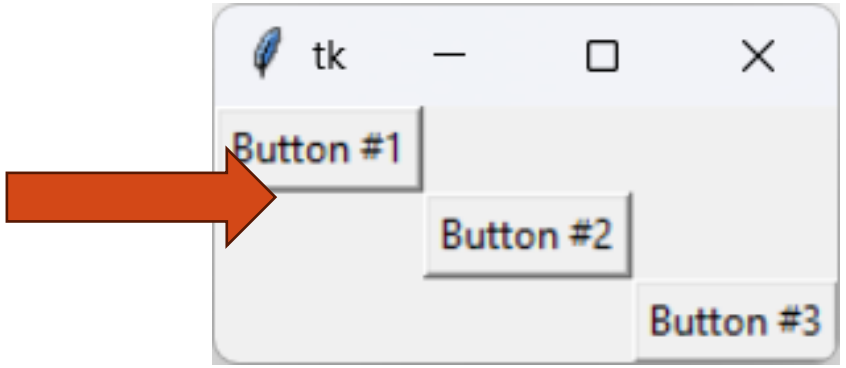
TKINTER ESSENTIALS

- **Gestores de geometrías (Geometry Managers)**
  - **Grid.**
  - **Principales parámetros:**
  - Comparte la responsabilidad con el desarrollador.
  - **Grid** distribuye el contenedor en una cuadrícula:
    - **column** = c. Indica la columna en la que se ubica el widget. Las columnas comienzan por 0. Si no se indica, toma el valor 0.
    - **row** = r. Indica la fila en la que se ubica el widget. Las filas comienzan por 0. Si no se indica, utiliza la primera fila libre comenzando por arriba.
    - **columnspan** = cs. Determina cuantas columnas adyacentes ocupa el widget. Por defecto es 0.
    - **rowspan** = rs. Igual que **columnspan**, pero con filas.
    - **sticky =** "n", "s", "e", "w", "ns", "ew" pega a la orientación indicada el componente. Utilizar conjuntamente con **rowspan** y **columnspan** para que se extiendan.

TKINTER ESSENTIALS

- **Gestores de geometrías (Geometry Managers)**
  - **Grid.**

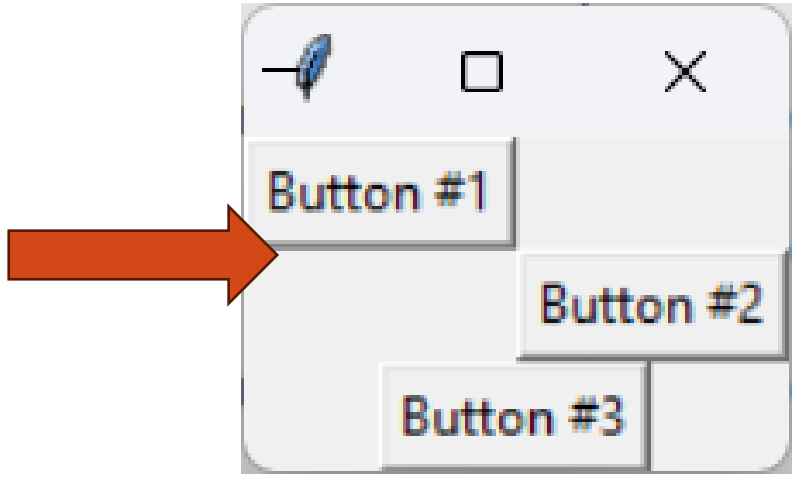
```
import tkinter as tk

window = tk.Tk()
button_1 = tk.Button(window, text="Button #1")
button_2 = tk.Button(window, text="Button #2")
button_3 = tk.Button(window, text="Button #3")
button_1.grid(row=0, column=0)
button_2.grid(row=1, column=1)
button_3.grid(column=2) # Por defecto en la row=2
window.mainloop()
```



TKINTER ESSENTIALS

- **Gestores de geometrías (Geometry Managers)**
  - **Grid. Atributo columnspan.**

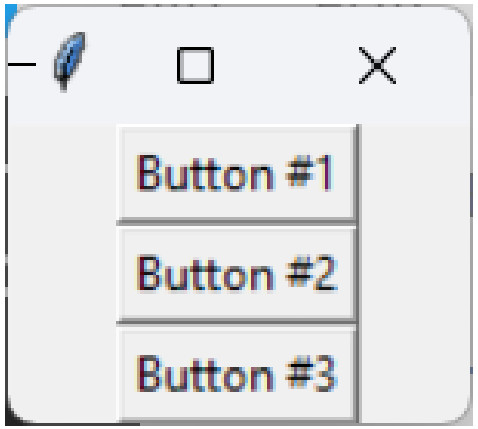
```
import tkinter as tk

window = tk.Tk()
button_1 = tk.Button(window, text="Button #1")
button_2 = tk.Button(window, text="Button #2")
button_3 = tk.Button(window, text="Button #3")
button_1.grid(row=0, column=0)
button_2.grid(row=1, column=1)
button_3.grid(row=2, column=0, columnspan=2)
window.mainloop()
```



TKINTER ESSENTIALS

- **Gestores de geometrías (Geometry Managers)**
  - **Pack.**
  - Por defecto, ubica cada widget en una fila.
  - Recomendado para prototipado, pero no como mejor opción.

```
import tkinter as tk

window = tk.Tk()
button_1 = tk.Button(window, text="Button #1")
button_2 = tk.Button(window, text="Button #2")
button_3 = tk.Button(window, text="Button #3")
button_1.pack()
button_2.pack()
button_3.pack()
window.mainloop()
```



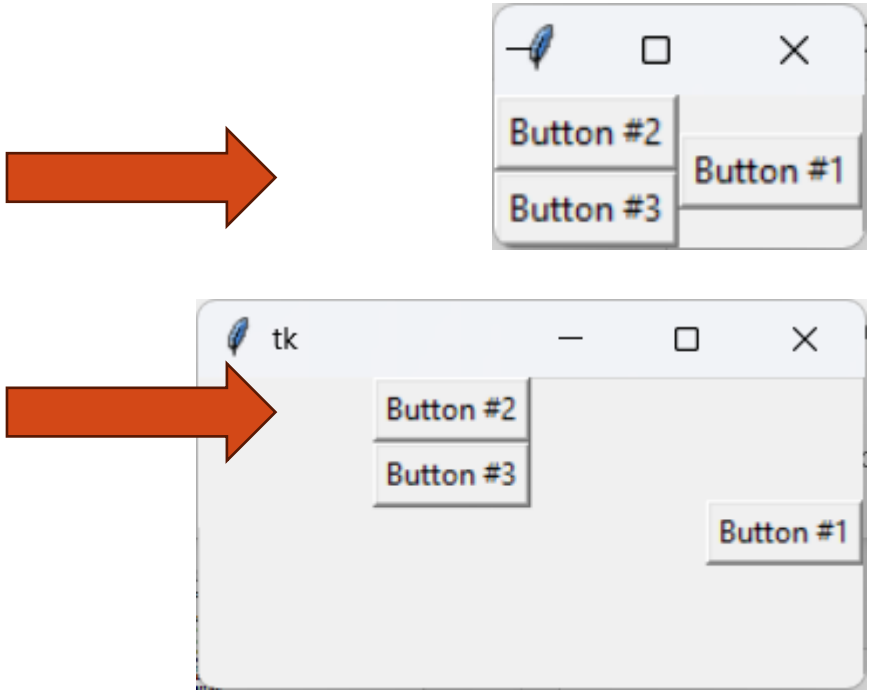
TKINTER ESSENTIALS

- **Gestores de geometrías (Geometry Managers)**
  - **Pack. Principales parámetros:**
  - **side** = s. Dirección en la que se ubica el widget. Valores posibles:
    - **TOP**
    - **BOTTOM**
    - **LEFT**
    - **RIGHT**
  - **fill** = f. Indica como expandir el widget. Valores posibles:
    - NONE. Valor por defecto. No expande el widget.
    - X. Expande horizontalmente.
    - Y. Expande verticalmente.
    - BOTH. Expande en las dos direcciones.
  - **Expand** = b. Indica si expandir o no el widget (True/False)

TKINTER ESSENTIALS

- **Gestores de geometrías (Geometry Managers)**
  - **Pack. Ejemplo atributo side:**

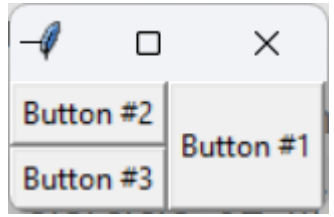
```
import tkinter as tk

window = tk.Tk()
button_1 = tk.Button(window, text="Button #1")
button_2 = tk.Button(window, text="Button #2")
button_3 = tk.Button(window, text="Button #3")
button_1.pack(side=tk.RIGHT)
button_2.pack()
button_3.pack()
window.mainloop()
```

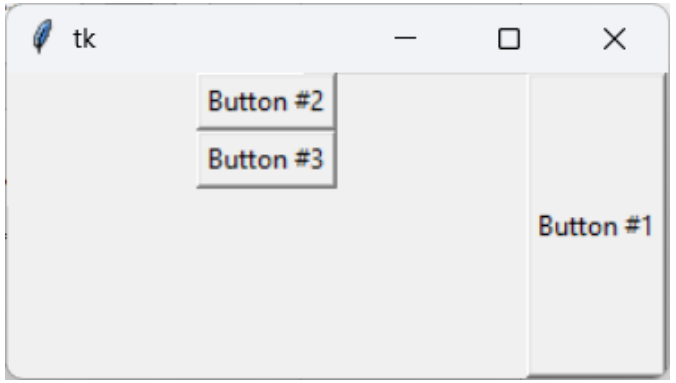


TKINTER ESSENTIALS

- **Gestores de geometrías (Geometry Managers)**
  - **Pack. Ejemplo atributo fill:**

```
import tkinter as tk

window = tk.Tk()
button_1 = tk.Button(window, text="Button #1")
button_2 = tk.Button(window, text="Button #2")
button_3 = tk.Button(window, text="Button #3")
button_1.pack(side=tk.RIGHT, fill=tk.Y)
button_2.pack()
button_3.pack()
window.mainloop()
```





TKINTER ESSENTIALS

Agregando colores

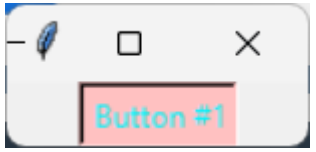
- La mayoría de los widgets pueden tener color.
- Python reconoce unos 750 colores predefinidos (pulsa aquí para ver la lista completa)

Named Color Chart

snow	deep sky blue	gold	seashell3	SlateBlue2	LightBlue3	SpringGreen2	DarkGoldenrod1	brown4	pink3	purple1	gray26	gray64
ghost white	sky blue	light goldenrod	seashell4	SlateBlue3	LightBlue4	SpringGreen3	DarkGoldenrod2	salmon1	pink4	purple2	gray27	gray65
white smoke	light sky blue	goldenrod	AntiqueWhite1	SlateBlue4	LightCyan2	SpringGreen4	DarkGoldenrod3	salmon2	LightPink1	purple3	gray28	gray66
gainsboro	steel blue	dark goldenrod	AntiqueWhite2	RoyalBlue1	LightCyan3	green2	DarkGoldenrod4	salmon3	LightPink2	purple4	gray29	gray67
floral white	light steel blue	rosy brown	AntiqueWhite3	RoyalBlue2	LightCyan4	green3	RosyBrown1	salmon4	LightPink3	MediumPurple1	gray30	gray68
old lace	light blue	indian red	AntiqueWhite4	RoyalBlue3	PaleTurquoise1	green4	RosyBrown2	LightSalmon2	LightPink4	MediumPurple2	gray31	gray69
linen	powder blue	saddle brown	bisque2	RoyalBlue4	PaleTurquoise2	chartreuse2	RosyBrown3	LightSalmon3	PaleVioletRed1	MediumPurple3	gray32	gray70
antique white	pale turquoise	sandy brown	bisque3	blue2	PaleTurquoise3	chartreuse3	RosyBrown4	LightSalmon4	PaleVioletRed2	MediumPurple4	gray33	gray71
papaya whip	dark turquoise	dark salmon	bisque4	blue3	PaleTurquoise4	chartreuse4	IndianRed1	orange2	PaleVioletRed3	thistle1	gray34	gray72
blanched almond	medium turquoise	salmon	PeachPuff2	DodgerBlue2	CadetBlue1	OliveDrab1	IndianRed2	orange3	PaleVioletRed4	thistle2	gray35	gray73
bisque	turquoise	light salmon	PeachPuff3	DodgerBlue3	CadetBlue2	OliveDrab2	IndianRed3	orange4	maroon1	thistle3	gray35	gray74
peach puff	cyan	orange	PeachPuff4	DodgerBlue4	CadetBlue3	OliveDrab4	IndianRed4	DarkOrange1	maroon2	thistle4	gray37	gray75
navajo	light cyan	dark orange	NavajoWhite2	SteelBlue1	CadetBlue4	DarkOliveGreen1	sienna1	DarkOrange2	maroon3	gray1	gray38	gray76
lemon chiffon	cadet blue	coral	NavajoWhite3	SteelBlue2	turquoise1	DarkOliveGreen2	sienna2	DarkOrange3	maroon4	gray2	gray39	gray77
mint cream	medium aquamarine	light coral	NavajoWhite4	SteelBlue3	turquoise2	DarkOliveGreen3	sienna3	DarkOrange4	VioletRed1	gray3	gray40	gray78
azure	aquamarine	tomato	LemonChiffon2	SteelBlue4	turquoise3	DarkOliveGreen4	sienna4	coral1	VioletRed2	gray4	gray42	gray79
alice blue	dark green	orange red	LemonChiffon3	DeepSkyBlue2	turquoise4	khaki1	burlywood1	coral2	VioletRed3	gray5	gray43	gray80
lavender	dark olive green	red	LemonChiffon4	DeepSkyBlue3	cyan2	khaki2	burlywood2	coral3	VioletRed4	gray6	gray44	gray81
lavender blush	dark sea green	hot pink	cornsilk2	DeepSkyBlue4	cyan3	khaki3	burlywood3	coral4	magenta2	gray7	gray45	gray82
misty rose	sea green	deep pink	cornsilk3	SkyBlue1	cyan4	khaki4	burlywood4	tomato2	magenta3	gray8	gray46	gray83
dark slate gray	medium sea green	pink	cornsilk4	SkyBlue2	DarkSlateGray1	LightGoldenrod1	wheat1	tomato3	magenta4	gray9	gray47	gray84
dim gray	light sea green	light pink	ivory2	SkyBlue3	DarkSlateGray2	LightGoldenrod2	wheat2	tomato4	orchid1	gray10	gray48	gray85
slate gray	pale green	pale violet red	ivory3	SkyBlue4	DarkSlateGray3	LightGoldenrod3	wheat3	OrangeRed2	orchid2	gray11	gray49	gray86
light slate gray	spring green	maroon	ivory4	LightSkyBlue1	DarkSlateGray4	LightGoldenrod4	wheat4	OrangeRed3	orchid3	gray12	gray50	gray87
gray	lawn green	medium violet red	honeydew2	LightSkyBlue2	aquamarine2	LightYellow2	tan1	OrangeRed4	orchid4	gray13	gray51	gray88
light grey	medium spring green	violet red	honeydew3	LightSkyBlue3	aquamarine4	LightYellow3	tan2	red2	plum1	gray14	gray52	gray89
midnight blue	green yellow	medium orchid	honeydew4	LightSkyBlue4	DarkSeaGreen1	LightYellow4	tan4	red3	plum2	gray15	gray53	gray90
navy	lime green	dark orchid	LavenderBlush2	SlateGray1	DarkSeaGreen2	yellow2	chocolate1	red4	plum3	gray16	gray54	gray91
cornflower blue	yellow green	dark violet	LavenderBlush3	SlateGray2	DarkSeaGreen3	yellow3	chocolate2	DeepPink2	plum4	gray17	gray55	gray92
dark slate blue	forest green	blue violet	LavenderBlush4	SlateGray3	DarkSeaGreen4	yellow4	chocolate3	DeepPink3	MediumOrchid1	gray18	gray56	gray93
slate blue	olive drab	purple	MistyRose2	SlateGray4	SeaGreen1	gold2	firebrick1	DeepPink4	MediumOrchid2	gray19	gray57	gray94
medium slate blue	dark khaki	medium purple	MistyRose3	LightSteelBlue1	SeaGreen2	gold3	firebrick2	HotPink1	MediumOrchid3	gray20	gray58	gray95
light slate blue	khaki	thistle	MistyRose4	LightSteelBlue2	SeaGreen3	gold4	firebrick3	HotPink2	MediumOrchid4	gray21	gray59	gray97
medium blue	pale goldenrod	snow2	azure2	LightSteelBlue3	PaleGreen1	goldenrod1	firebrick4	HotPink3	DarkOrchid1	gray22	gray60	gray98
royal blue	light goldenrod yellow	snow3	azure3	LightSteelBlue4	PaleGreen2	goldenrod2	brown1	HotPink4	DarkOrchid2	gray23	gray61	gray99
blue	light yellow	snow4	azure4	LightBlue1	PaleGreen3	goldenrod3	brown2	pink1	DarkOrchid3	gray24	gray62	
dodger blue	yellow	seashell2	SlateBlue1	LightBlue2	PaleGreen4	goldenrod4	brown3	pink2	DarkOrchid4	gray25	gray63	

TKINTER ESSENTIALS

- **Colores**
  - Atributos:
    - bg (background).
    - fg (foreground)
    - activebackground
    - activeforeground




```
import tkinter as tk

window = tk.Tk()
button = tk.Button(window, text="Button #1", bg="red", fg="yellow",
        activebackground="RosyBrown1", activeforeground="turquoise1")

button.pack()
window.mainloop()
```

TKINTER ESSENTIALS

- **Colores:**
  - Se pueden asignar indicando el nombre en inglés
  - Se pueden asignar indicando el color utilizando el modelo de color RGB.
  - Se indican mediante una cadena con el símbolo de almohadilla y el valor de cada color primario en hexadecimal.

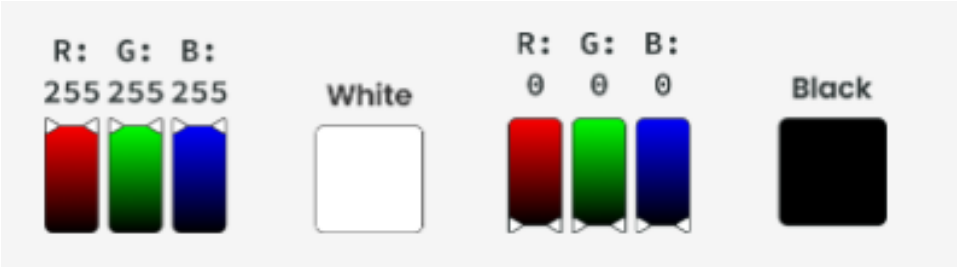
```
import tkinter as tk

window = tk.Tk()
button = tk.Button(window, text="Button #1",
bg="#00bb00", fg="#cccccc")
button.pack()
window.mainloop()
```

TKINTER ESSENTIALS

- **Colores. Modelo de color RGB:**
  - RGB determina un color como la combinación los tres colores primarios (modelo de color RGB, uno de los modelos de color **aditivo**).
  - Un color RGB es una combinación de tres números entre 0 y 255 correspondientes a los colores Red, Green y Blue. Proporcionan un total de 16.777.216 (2^8^3) colores posibles (el ojo humano distingue aproximadamente 7.000.000 de colores distintos).

TKINTER ESSENTIALS

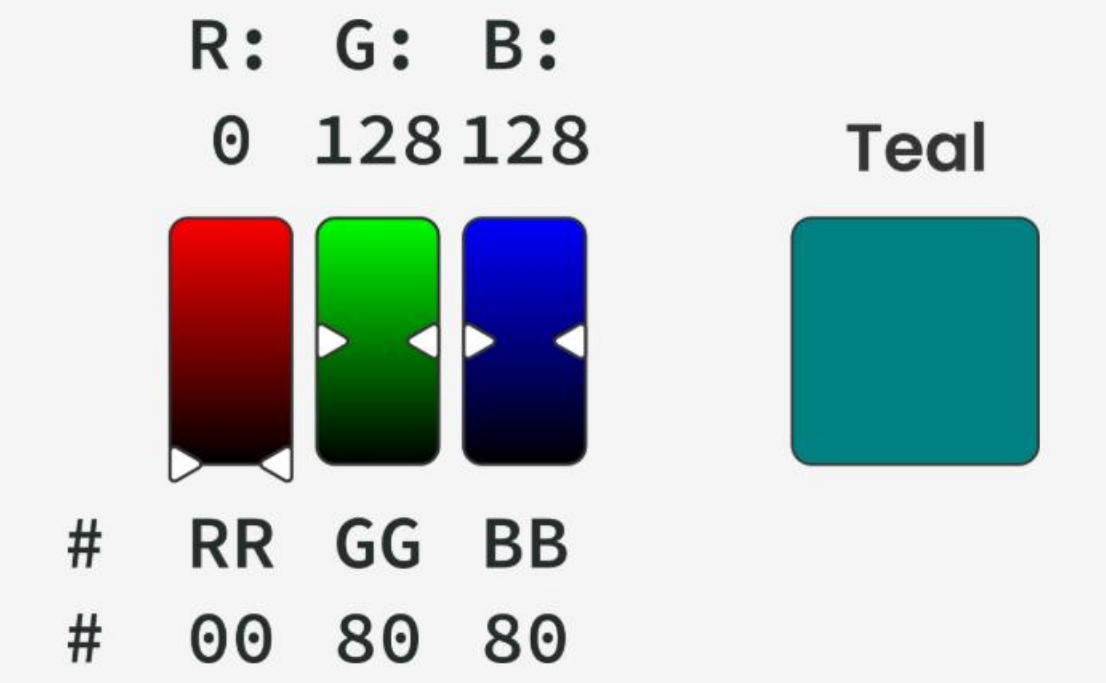
- **Colores. Modelo de color RGB:**
  - Si todos los componentes tienen valor 0 → Color negro
  - Si todos los componentes tienen valor 255 → Color blanco
  - Si todos los componentes tienen el mismo valor → Tono de gris



  - Si un componente está en 255 y el resto en 0, color primario.

TKINTER ESSENTIALS

- **Colores. Modelo de color RGB:**
  - Resto de combinaciones producen el color resultante.



TKINTER ESSENTIALS

- **Otros componentes:**
  - **Label**. Etiqueta de texto no editable. No clicable (ver nota).
  - **Frame**. Permite agrupar widgets. No clicable (ver nota).
  - **StringVar, IntVar, DoubleVar, BooleanVar**. Tipos de variable para la comunicación entre widgets (observables).
    - No admite asignación directa → Utilización de métodos set.
  - **Checkbutton**. Almacena el valor en un atributo llamado **variable** al que se le asigna un objeto que recoja el valor de la selección (BooleanVar o IntVar). La relación entre el checkbox y la variable es direccional: si se cambia el valor, se cambia el estado y viceversa → '*two-way binding*'

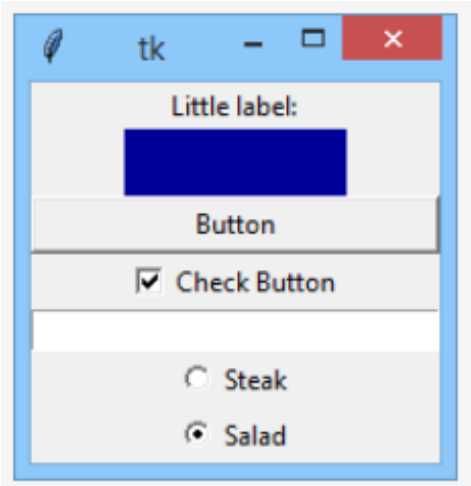
- **NOTA**: Cuando se afirma en este contexto que no es clicable, significa que no tiene parámetro **command**. Existen la posibilidad de asignar eventos mediante el método **bind*(evento, función callback)*** .

TKINTER ESSENTIALS

- **Otros componentes:**
  - **Entry**. Widget para recoger la entrada de texto del usuario.
  - **Radiobutton**. Selector excluyente. Tiene dos parámetros específicos:
    - **variable**: relaciona el valor del componente con una variable tk.
    - **value**: permite indicar el valor que tendrá **variable** cuando el elemento esté seleccionado.

TKINTER ESSENTIALS

- **Creación de una aplicación simple:**



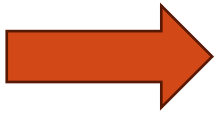

```
import tkinter as tk

window = tk.Tk()

label = tk.Label(window, text="Little label:")
label.pack()

frame = tk.Frame(window, height=30, width=100, bg="#000099")
frame.pack()

button = tk.Button(window, text ="Button")
button.pack(fill=tk.X)

switch = tk.IntVar()
switch.set(1)

checkbutton = tk.Checkbutton(window, text="Check Button", variable=switch)
checkbutton.pack()

entry = tk.Entry(window, width=30)
entry.pack()

radiobutton_1 = tk.Radiobutton(window, text="Steak", variable=switch, value=0)
radiobutton_1.pack()
radiobutton_2 = tk.Radiobutton(window, text="Salad", variable=switch, value=1)
radiobutton_2.pack()

window.mainloop()
```

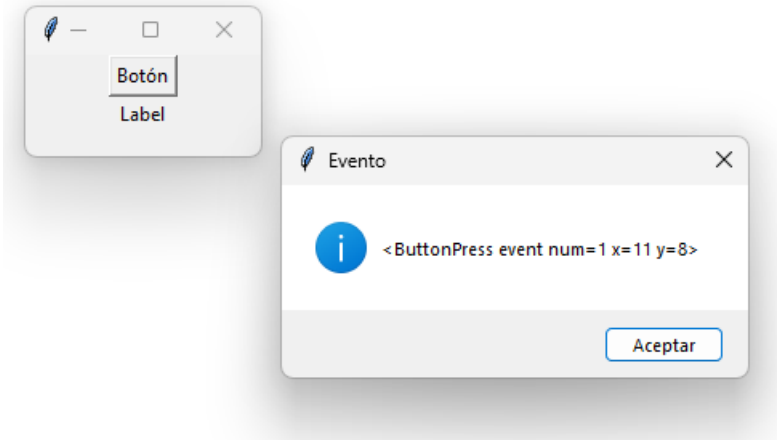
TKINTER ESSENTIALS

Manejo de eventos

- No todos los componentes disponen del atributo **command**
- Existen más eventos que el relacionado con **command**
- El método **bind** permite asignar un comportamiento a evento qué ha ocurrido sobre un widget. Existe el método **unbind**.
- La función callback recibe un objeto con información sobre el evento a través de un objeto de la clase **Event** (no así con **command**).
- El método **bind_all** permite asignar un comportamiento a todos los widgets.

```
import tkinter as tk
from tkinter import messagebox

def clicked(event=None):
    messagebox.showinfo("Evento", event)

window = tk.Tk()
button_1 = tk.Button(window, text="Botón", command=clicked)
label_1 = tk.Label(window, text="Label")
button_1.pack()
label_1.pack()
label_1.bind("<Button-1>", clicked)
window.mainloop()
```





TKINTER ESSENTIALS

- **Manejo de eventos. Eventos**

Event name	Event role
`<Button-1>`	Single left-click (if your mouse is configured for a right-handed user)
`<Button-2>`	Single middle-click
`<Button-3>`	Single right-click
`<ButtonRelease-1>`	Left mouse button release Note: there are also events named `<ButtonRelease-2>` and `<ButtonRelease-3>`
`<DoubleButton-1>`	Double left-click Note: there are also events named `<DoubleButton-2>` and `<DoubleButton-3>` Note again: the `<Button-1>` event is a part of `<DoubleButton-1>` too; if you assign a callback to `<Button-1>`, it will be launched, too!

TKINTER ESSENTIALS

- **Manejo de eventos. Eventos**

Event name	Event role
`<Enter>`	Mouse cursor appears over the widget
`<Leave>`	Mouse cursor leaves the widget area
`<Focus-In>`	The widget gains the focus
`<Focus-Out>`	The widget loses the focus
`<Return>`	The user presses the *Enter/Return* key
`<Key>`	The user presses **any** key

TKINTER ESSENTIALS

- **Manejo de eventos. Eventos**

Event name	Event role
`x`	The user presses *x* key (*x* can be neither a space nor the < key)
`<space>`	The user presses the *spacebar*
`<less>`	The user presses the < key
`<Cancel>`	The user presses the key/keys used by the current OS to stop the program (e.g., *Ctrl-C* or *Ctrl-Break*)
`<BackSpace>`	The user presses the *Backspace* key
`<Tab>`	The user presses **Tab** key

TKINTER ESSENTIALS

- **Manejo de eventos. Eventos**

Event name	Event role
`<Shift_L>`	The user presses one of the *Shift keys*
`<Control_L>`	The user presses one of the *Control keys*
`<Alt_L>`	The user presses one of the *Alt keys*
`<Pause>`	The user presses the *Pause key*
`<Caps_Lock>`	The user presses the *Caps Lock key*
`<Esc>`	The user presses the *Escape keys*

TKINTER ESSENTIALS

- **Manejo de eventos. Eventos**

Event name	Event role
`<Prior>`	The *Page Up* key
`<Next>`	The *Page Down* key
`<End>`	The *End* key
`<Home>`	The *Home* key
`<Left>` `<Right>` `<Up>` `<Down>`	*Cursor* (arrows) keys
`<Num_Lock>` `<Scroll_Lock>`	The two **Lock** keys
`<Shift-x>` `<Alt-x>` `<Control-x>`	The *x* key has been pressed along with any of the *Shift*, *Alt*, or *Control* keys

TKINTER ESSENTIALS

- **Manejo de eventos. Clase Event.**
  - Contiene propiedades con información sobre el evento.
  - No todos los eventos tienen información en las mismas propiedades.

Property name	Property role
`widget`	The widget's **object** (not the widget's name!) to which the event is addressed
`<x>` `<y>`	The mouse cursor's **coordinates** at the moment of the event's occurrence (both coordinates are counted relative to the target widget)
`<x_root>` `<y_root>`	As above, but relative to the **screen**
`<char>`	The pressed key **character code** (only for keyboard events)
`<keysym>`	The pressed key symbol (only for keyboard events) The full list of all recognized key symbols is presented here: https://www.tcl.tk/man/tcl8.4/TkCmd/keysyms.htm
`<keycode>`	The pressed key **numerical** code (only for keyboard events) Don't confuse this with `char`, which is the ASCII/UNICODE code of the character bound to the key
`<num>`	The number of the clicked mouse **button** (only for mouse events)
`<type>`	The event's type

TKINTER ESSENTIALS

- **Manejo de eventos. Desvincular eventos.**
  - Desvincular un evento asociado con **bind** a través de **unbind**.
  - Desvincular un evento asociando con **command** a través de **config**.
    - El método **config** permite modificar el valor de un atributo de un widget, indicando propiedad y valor.
    - *nombre_widget.*config(command=lambda:None)

  - El método **bind_all** vincula un evento a todos los widgets.

```
window.bind_all("<Button-1>", funcion_callback)
```

  - El método **unbind_all** desvincula un evento de todos los widgets.

```
window.unbind_all("<Button-1>")
```

TKINTER ESSENTIALS

- **Manejo de eventos. Desvincular eventos.**

```
import tkinter as tk
from tkinter import messagebox

def clicked(event=None):
    messagebox.showinfo("Evento", event)
    if (event):
        label_1.unbind("<Button-1>")
    else:
        button_1.config(command=lambda:None)

window = tk.Tk()
button_1 = tk.Button(window, text="Botón", command=clicked)
label_1 = tk.Label(window, text="Label")
button_1.pack()
label_1.pack()
label_1.bind("<Button-1>", clicked)
window.mainloop()
```

TKINTER ESSENTIALS

- **Propiedades de los widgets. Manejo.**
  - **Opción 1**. Manejando el widget como un diccionario.
    - Utiliza el nombre de la propiedad como clave:

```
state = button["text"]
if state == "ON":
    state = "OFF"
else:
    state = "ON"
    button["text"] = state
```

TKINTER ESSENTIALS

- **Propiedades de los widgets. Manejo.**
  - **Opción 2**. Utilizando los métodos **cget** y **config.**

```
state = button.cget("text")
if state == "ON":
    state = "OFF"
else:
    state = "ON"
button.config(text=state)
```

TKINTER ESSENTIALS

- **Propiedades de los widgets. Fuentes de letra.**
  - Se indica en la propiedad **font**.
  - Utiliza tuplas de dos o tres elementos para asignar la fuente:
    - ("font_family_name", "font_size")
    - ("font_family_name", "font_size", "font_style")
  - **font_size** contiene el número de puntos, pero debe ser una cadena.
  - **font_style** puede tomar los siguientes valores:
    - "**bold**"
    - "**italic**"
    - "**underline**"
    - "**overstrike**"
  - Ejemplos:
    - font=("Arial", "16", "bold"))
    - font=("Times", "12")

TKINTER ESSENTIALS

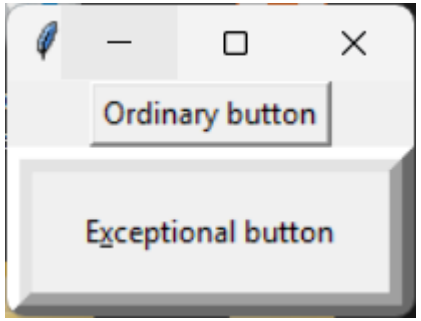
- **Propiedades de los widgets. Tamaños.**
  - Propiedades:

Widget property name	Property role
`borderwidth`	The width of the **3D-frame** surrounding some widgets (e.g., `Button`)
`highlightthickness`	The width of the additional frame drawn around the widget when it gains the focus
`padx` `pady`	The width/height of an additional empty space/margin around the widget
`wraplength`	If the text filling the widget becomes longer than this property's value, it will be wrapped (possibly more than once)
`height`	The height of the widget
`underline`	The index of the character inside the widget's text, which should be presented as underlined or `-1` otherwise (the underlined letter/digit can be used as a shortcut key, but it needs a specialized callback to work – no automation here, sorry)
`width`	The width of the widget

TKINTER ESSENTIALS

- **Propiedades de los widgets. Tamaños.**
  - Ejemplo:

```
import tkinter as tk


window = tk.Tk()
button_1 = tk.Button(window, text="Ordinary button");
button_1.pack()
button_2 = tk.Button(window, text="Exceptional button")
button_2.pack()
button_2["borderwidth"] = 10
button_2["highlightthickness"] = 10
button_2["padx"] = 10
button_2["pady"] = 5
button_2["underline"] = 1
window.mainloop()
```



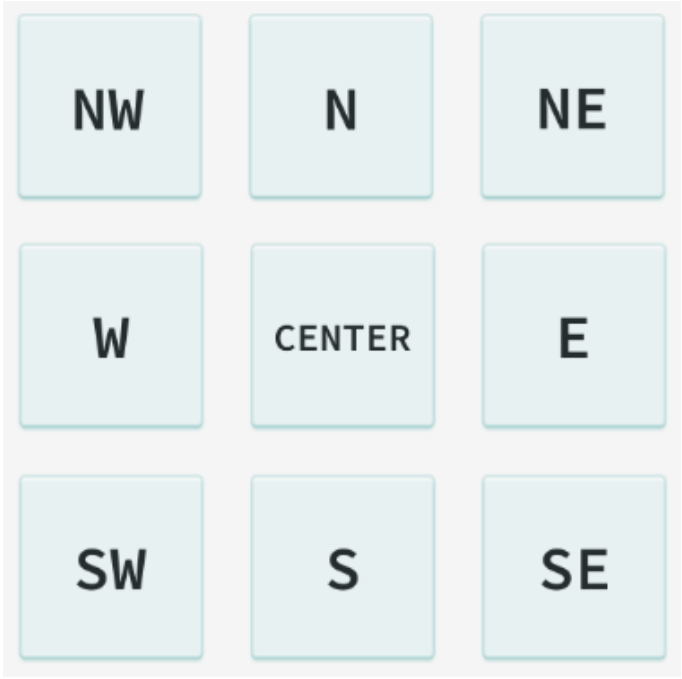
TKINTER ESSENTIALS

- **Propiedades de los widgets. Propiedades de color.**
  - Propiedades:

Widget property name	Property role
`background` `bg`	The color of the widget's background (you can freely use either of these two forms)
`foreground` `fg`	The color of the widget's foreground (note: it can mean different things in different widgets; in general, it's used to specify text color)
`activeforeground` `activebackground`	Like `bg` and fg but used when the widget becomes active
`disabledforeground`	The width of the widget

TKINTER ESSENTIALS

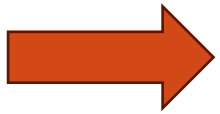
- **Propiedades de los widgets. anchor.**
  - Fija una posición de referencia (*anchor* o ancla) para fijar el contenido del componente.
  - Proporciona 9 posibilidades.

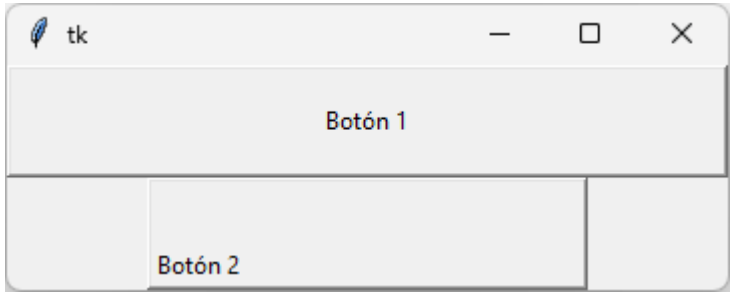


TKINTER ESSENTIALS

- **Propiedades de los widgets. anchor.**
  - Ejemplo.

```
import tkinter as tk

window = tk.Tk()
button_1 = tk.Button(window, text="Botón 1");
button_1["anchor"] = tk.CENTER
button_1["width"] = 50  # puntos
button_1["height"] = 3  # filas
button_1.pack()
button_2 = tk.Button(window, text="Botón 2")
button_2["anchor"] = tk.SW
button_2["width"] = 30
button_2["height"] = 3
button_2.pack()
window.mainloop()
```





TKINTER ESSENTIALS

- **Propiedades de los widgets. cursor.**
  - Ejemplo.

```
import tkinter as tk

window = tk.Tk()
label_1 = tk.Label(window, height=3, text="arrow", cursor="arrow")
label_1.pack()
label_2 = tk.Label(window, height=3, text="clock", cursor="clock")
label_2.pack()
label_3 = tk.Label(window, height=3, text="heart", cursor="heart")
label_3.pack()
window.mainloop()
```

TKINTER ESSENTIALS

- **Propiedades de los widgets. Métodos.**
  - **after** → Invoca a una función callback UNA ÚNICA VEZ tras esperar un número de ms
  - **after_cancel** → Cancela la invocación

```
import tkinter as tk


def blink():
    global is_white
    if is_white:
        color = 'black'
    else:
        color = 'white'
    is_white = not is_white
    frame.config(bg=color)
    frame.after(500, blink)


is_white = True
window = tk.Tk()
frame = tk.Frame(window, width=200, height=100, bg='white')
frame.after(500, blink)
frame.pack()
window.mainloop()
```

TKINTER ESSENTIALS

- **Propiedades de los widgets. Métodos.**
  - **destroy** → Destruye un widget y todos sus descendientes (es recursivo).
    - Aplicado a la ventana principal (bucle principal), finaliza la aplicación.
  - **focus_get**:
    - Indica que widget tiene el foco o None.
    - Se puede invocar desde cualquier widget.
  - **focus_set**:
    - Asigna el foco al widget desde el que se ha invocado.

TKINTER ESSENTIALS

- **Propiedades de los widgets. Métodos.**
  - **focus_set**: Ejemplo

```
import tkinter as tk


def flip_focus():
    if window.focus_get() is button_1:
        button_2.focus_set()
    else:
        button_1.focus_set()
    window.after(1000, flip_focus)


window = tk.Tk()
button_1 = tk.Button(window, text="First")
button_1.pack()
button_2 = tk.Button(window, text="Second")
button_2.pack()
window.after(1000, flip_focus)
window.mainloop()
```

TKINTER ESSENTIALS

- **Variables observables.**
  - Variables visibles por los componentes de la GUI.
  - Deben declararse explícitamente.
  - Tienen tipo.
  - Sólo se pueden crear después de inicializar la ventana principal.
  - Se puede acceder a sus valores con los métodos **get** y **set.**
  - Tipos de observables:
    - BooleanVar. Por defecto, un **bool** con valor False.
    - DoubleVar. Por defecto, un **float** con valor 0.0
    - IntVar. Por defecto, un **int** con valor 0.
    - StringVar. Por defecto, un **str** con valor “”.

TKINTER ESSENTIALS

- **Variables observables y observadores.**
  - Una variable **observable** se puede asociar con un **observador (observer)**.
  - Un **observer** es una función que se va a invocar cuando la variable **observable** cambie de estado. **Tiene una estructura concreta.**
  - Se pueden utilizar como sustituto de la propiedad **text** asignando la variable a la propiedad **textvariable.**
  - El número de observadores es ilimitado.
  - Se asignan mediante el método **trace_add (**~~trace~~ está obsoleto) de la variable observable, en el que se indica el tipo de cambio que se quiere "observar":
    - "read"(~~'r'~~ está obsoleto) : las lecturas de la variable (accediendo a su valor mediante get())
    - "write"(~~'r'~~ está obsoleto) : las escrituras de la variable (cambiando su valor mediante set())
    - "unset" (~~'u'~~ está obsoleto): la eliminación de la variable (eliminando el objeto mediante del)
  - Devuelve un identificador que se necesitará para eliminar el observador mediante el método **trace_remove() (**~~trace_vdelete~~ está obsoleto**)**
  - Sintaxis:

```
obsid = variable.trace_add(trace_mode, observer)
```

TKINTER ESSENTIALS

- **Observadores. Estructura**
  - Estructura del **observer**.
    - **def funcion_observer(vid, ix, act):**
      - **vid:** identificador o nombre de la variable observable.
      - **ix:** índice o cadena vacía.
      - **act:** una cadena indicando qué ocurrió en el observable (‘read’, ‘write’ o ‘unset’)
      - Si no se van a utilizar, **se pueden agrupar todos en *args.**

TKINTER ESSENTIALS

- **Observadores. Ejemplo:**

```
import tkinter as tk

def r_observer(*args):
    print("Reading")

def w_observer(*args):
    print("Writing")

ventana = tk.Tk()
variable = tk.StringVar()
boton_1 = tk.Button(ventana, text="Leer", command=lambda: variable.get())
boton_2 = tk.Button(ventana, text="Escribir", command=lambda: variable.set("Hola"))
boton_3 = tk.Button(ventana, text="Dejar de observar leer", command=lambda: variable.trace_remove("read", r_obsid))
boton_1.pack()
boton_2.pack()
boton_3.pack()
r_obsid = variable.trace_add("read", r_observer)
w_obsid = variable.trace_add("write", w_observer)

ventana.mainloop()
```