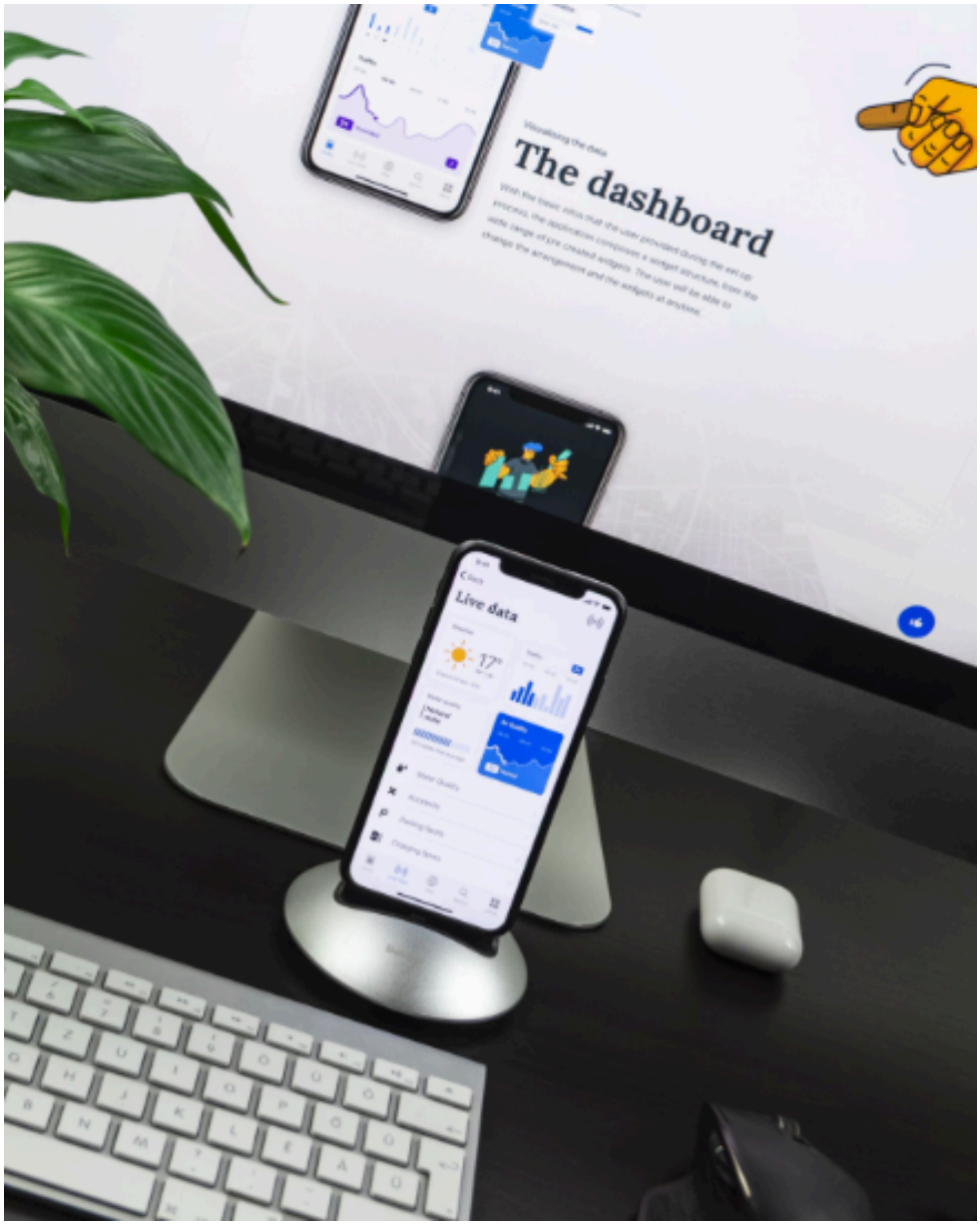
TIC

Lección 2: Interfaces con Tkinter



Tiempo de ejecución: 4 horas
Planteamiento de la sesión

Materiales

- Interfaces gráficas en Python con Tkinter – Adictos al trabajo
- pack manual page – Tk Built-In Commands
- tkinter — Python interface to Tcl/Tk
- Introducción a Tcl/Tk (tkinter) – Recursos Python
- Apariencia y estilos de los controles en Tcl/Tk (tkinter) – Recursos Python
- tkinter — Interface de Python para Tcl/Tk
- Posicionar elementos en Tcl/Tk (tkinter) – Recursos Python
- Botón (Button) en Tcl/Tk (tkinter) – Recursos Python
- Caja de texto (Entry) en Tcl/Tk (tkinter) – Recursos Python

Introducción

Tkinter es el paquete más utilizado para crear interfaces gráficas en Python. Es una capa orientada a objetos basada en Tcl (sencillo y versátil lenguaje de programación open-source) y Tk (la herramienta GUI estándar para Tcl).

Es una herramienta para desarrollar aplicaciones de escritorio multiplataforma, esto es, aplicaciones nativas con una interfaz gráfica para sistemas operativos Windows, Linux, Mac y otros. Técnicamente, Tk es una biblioteca de código abierto escrita en C y desarrollada en sus orígenes para el lenguaje de programación Tcl; de ahí que usualmente nos refiramos a ella como Tcl/Tk. Desde sus primeras versiones Python incluye en su biblioteca o librería estándar el módulo tkinter, que permite interactuar con Tk para desarrollar aplicaciones de escritorio en Python.

La curva de aprendizaje de Tk es relativamente pequeña si la comparamos con otras bibliotecas del rubro (como Qt), de modo que cualquier programador con una mínima base de Python puede comenzar rápidamente a crear aplicaciones gráficas profesionales y luego distribuirlas vía herramientas como cx_Freeze o PyInstaller, que se integran muy bien con Tk.

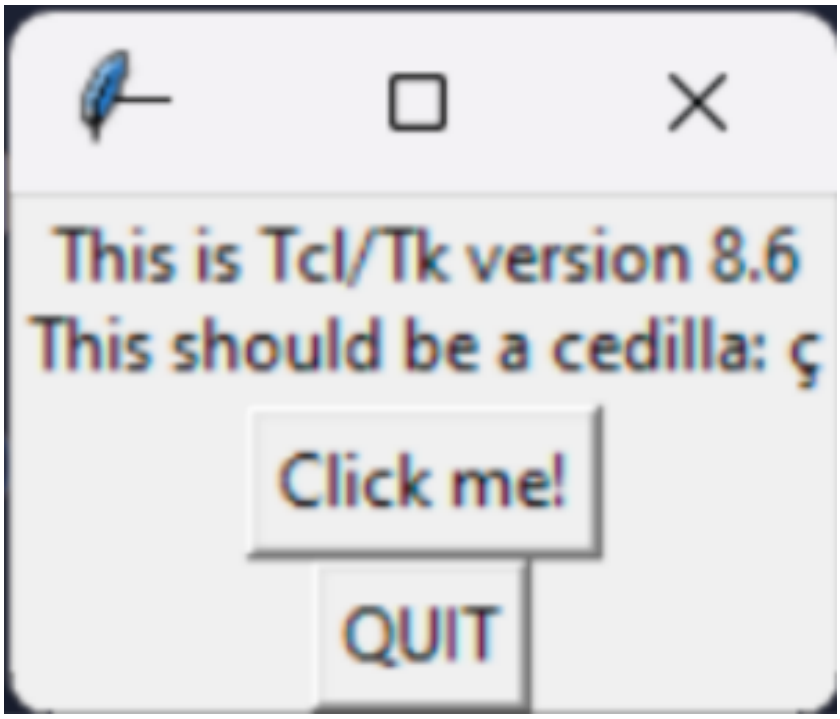
Ejercitación

Vamos a comenzar conociendo un poco de esta herramienta con un ejercicio sencillo, en donde vamos a crear un sistema que ingresado un valor de temperatura en Celsius, obtengamos su equivalente en Kelvin y Fahrenheit.

Este mismo principio lo van a utilizar ustedes para otra ejercitación en donde vamos a convertir medidas de distancia como ingresado un valor en kilómetros devolver su equivalente en millas, metros, pies, etc.

Lo primero es conocer que la herramienta viene incluida dentro de Python, por lo que no es necesario instalar nada por ahora.

Para verificar esto, vamos a ejecutar python –m tkinter desde la línea de comandos debería abrir una ventana que muestre una interfaz Tk simple para saber si tkinter está instalado correctamente en su sistema. También muestra qué versión de Tcl/Tk está instalada para que pueda leer la documentación de Tcl/Tk específica de esa versión.



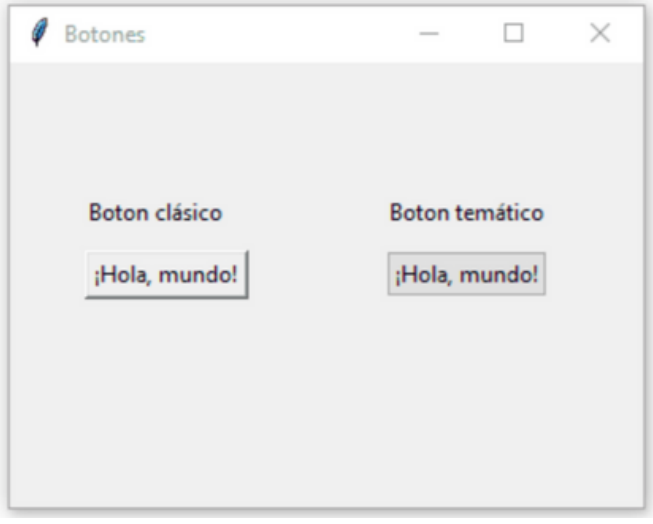
Vamos a crear entonces un nuevo proyecto con su respectivo archivo main.py y comenzamos.

El primer paso para cualquier aplicación de Tk es importar los módulos correspondientes:

```
import tkinter as tk
from tkinter import ttk
```

Aquí importamos el módulo principal tkinter abreviado como tk, una convención habitual entre los programadores de Python. En segundo lugar, importamos el módulo ttk que se encuentra dentro de tkinter. Estaremos utilizando objetos que están dentro de ambos módulos.

El hecho de que existan dos métodos distintos para modificar la estética de la interfaz radica en que en Tcl/Tk hay dos tipos de controles (widgets) que, un poco arbitrariamente, llamaremos controles clásicos y controles temáticos. Los controles clásicos son los controles originales de Tk, a los cuales desde Python accedemos vía el módulo tkinter (que generalmente se abrevia tk). Los controles temáticos se introdujeron en la versión 8.5 de Tk y están contenidos en el submódulo de Python ttk. La diferencia entre los dos tipos de controles es principalmente estética: los controles temáticos tienen una apariencia más moderna e incluyen un sistema de personalización de esa apariencia mejor que el sistema tradicional. Véase la diferencia entre un botón creado a partir de la clase tk.Button (izquierda) y otro a partir de ttk.Button (derecha).



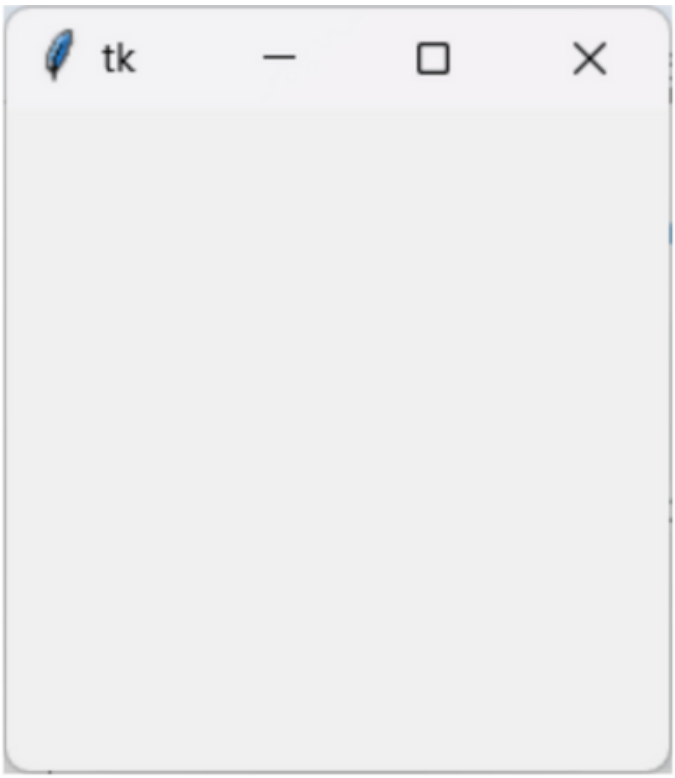
Fuente: Apariencia y estilos de los controles en Tcl/Tk (tkinter) – Recursos Python

Cuando un control está disponible tanto en su versión clásica como temática, siempre conviene la temática, puede ver un listado en el artículo Apariencia y estilos de los controles en Tcl/Tk (tkinter) – Recursos Python

El segundo paso fundamental es crear la ventana principal. Todas las aplicaciones de Tk tendrán una ventana principal, y eventualmente algunas otras ventanas secundarias.

```
import tkinter as tk
from tkinter import ttk

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

Debería verse así:



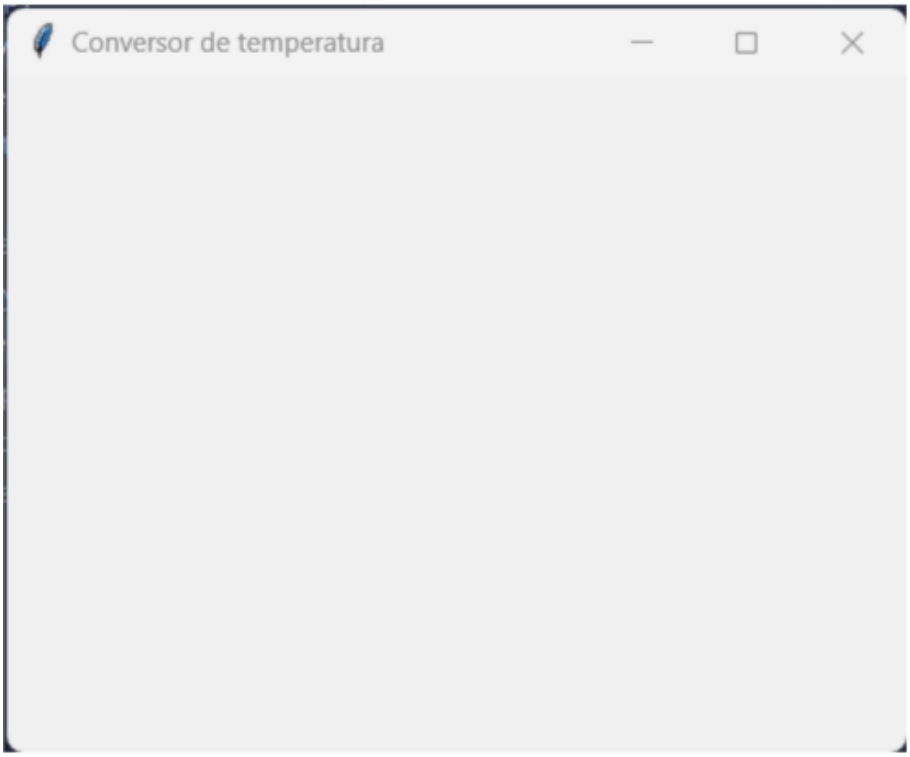
En la primera línea creamos una instancia de Tk, que se ocupa de crear la ventana principal y de iniciar internamente un intérprete de Tcl y Tk. Lo que aquí llamamos ventana comúnmente por convención entre programadores se conoce como root dado que es la raíz de donde inicia el resto de la interfaz.

La segunda línea ejecuta el método mainloop(), que es el bucle principal del programa. Todas las aplicaciones de escritorio trabajan con un bucle principal que se ocupa de gestionar los eventos de la interfaz gráfica. El bucle principal se está ejecutando constantemente (pues, justamente, es un bucle) y una de sus tareas principales es «dibujar» la ventana en la pantalla, por lo cual el método mainloop() solo finaliza cuando se cierra la última ventana de nuestra aplicación. Por lo general, ventana.mainloop() será la última línea de nuestro código.

Ahora vamos a dar un título y un tamaño a la ventana, vía los métodos title() y config().

```
import tkinter as tk
from tkinter import ttk

ventana = tk.Tk()
ventana.title("Conversor de temperatura")
ventana.config(width=400, height=300)
ventana.mainloop()
```

Para una nueva ventana como la siguiente:



Width y height son definidos en píxeles.

Ahora está lista para comenzar a llenar nuestra ventanas con los componentes de nuestro aplicativo, estos componentes se conocen como control o widget. Aquí es donde entra un elemento primordial que debemos definir, vamos a trabajar orientado a objetos o no, pues bien, la forma orientada a objetos es interesante y más útil para aplicaciones grandes y complejas lo que permite mantener un orden jerárquico y escalar nuestra aplicación fácilmente.

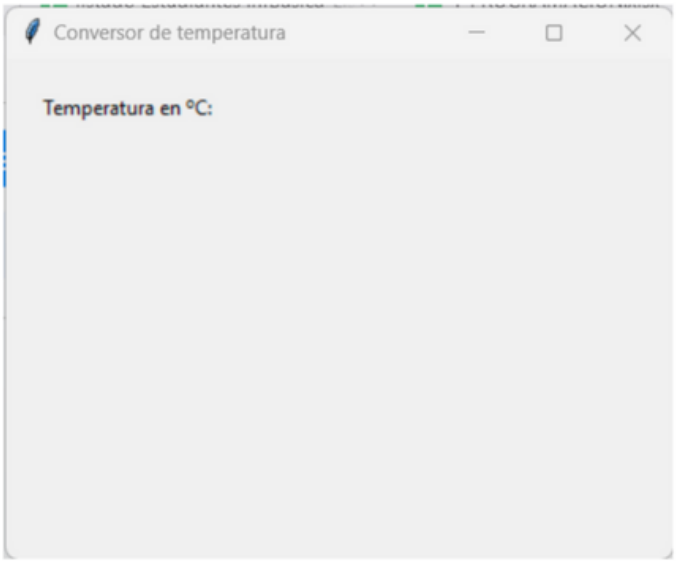
Sin embargo la forma no orientada a objetos puede ser útil para aplicaciones pequeñas y medianas con pocos elementos, para el ejercicio de nuestro aplicativo, vamos a revisar los dos aspectos y tomaremos una decisión para futuros aplicativos.

Dentro de la documentación encontramos un listado de los controles o widgets que podemos utilizar tkinter.tk — Tk widgets temáticos — documentación de Python - 3.12.3 o acá un listado Tk Commands, versión 8.6.14 o nuevamente el artículo que nos lista el widget tanto en clásico como temático, Apariencia y estilos de los controles en Tcl/Tk (tkinter) - Recursos Python

Enfoque Procedimental

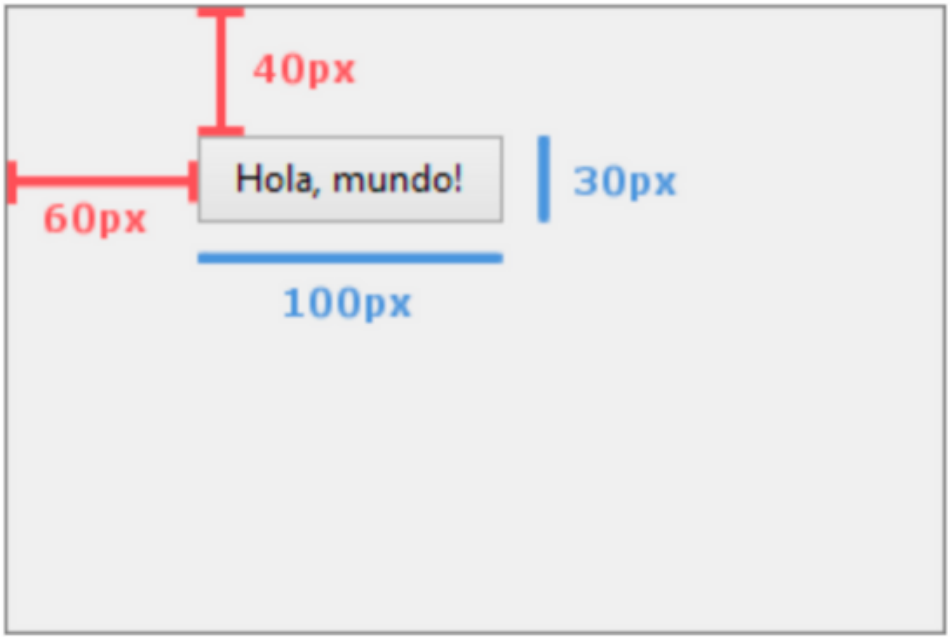
Sin embargo, vamos a ir viendo de forma práctica la utilización de algunos de ellos. Vamos a comenzar con una etiqueta o Label()

```
import tkinter as tk
from tkinter import ttk

ventana = tk.Tk()
ventana.title("Conversor de temperatura")
ventana.config(width=400, height=300)

etiqueta temp celsius = ttk.Label(text="Temperatura en oC:")
etiqueta_temp_celsius.place(x=20, y=20)

ventana.mainloop()
```



Para introducir un control en la interfaz, primero debemos crear una instancia de la clase correspondiente (ttk.Label, en este caso), asignársela a una variable (etiqueta_temp_celsius) y ubicarla en algún lugar de la ventana vía el método place(). Este método requiere que indiquemos de forma absoluta la posición del control en la ventana, esto es, especificando la posición en las coordenadas X e Y. Opcionalmente podemos pasar los argumentos width y height para asignarle al control un ancho y un alto fijos; de lo contrario, Tk provee un tamaño por defecto.

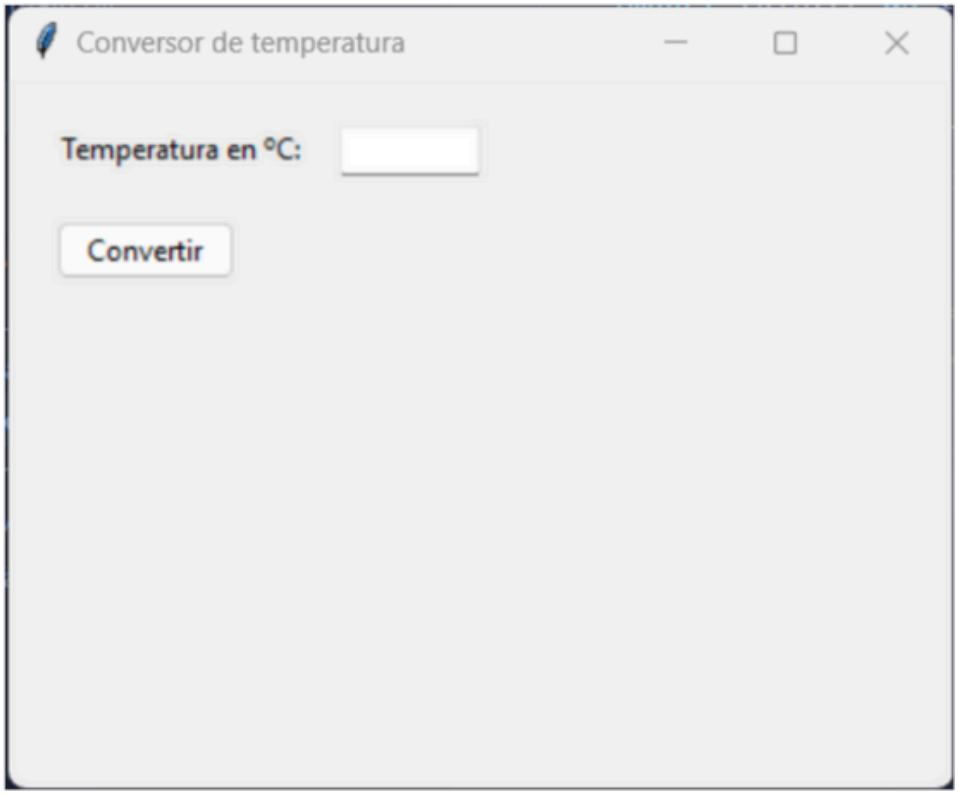
self.button.place(x=60, y=40, width=100, height=30)



Debajo de la creación de la etiqueta, agreguemos la caja de texto para introducir la temperatura y el botón para realizar la conversión. Para ello usaremos dos clases nuevas: ttk.Entry y ttk.Button.

```
import tkinter as tk
from tkinter import ttk
```

```
ventana = tk.Tk()
ventana.title("Conversor de temperatura")
ventana.config(width=400, height=300)

etiqueta_temp_celsius = ttk.Label(text="Temperatura en oC:")
etiqueta_temp_celsius.place(x=20, y=20)

caja_temp_celsius = ttk.Entry()
caja_temp_celsius.place(x=140, y=20, width=60)

boton_convertir = ttk.Button(text="Convertir")
boton_convertir.place(x=20, y=60)

ventana.mainloop()
```

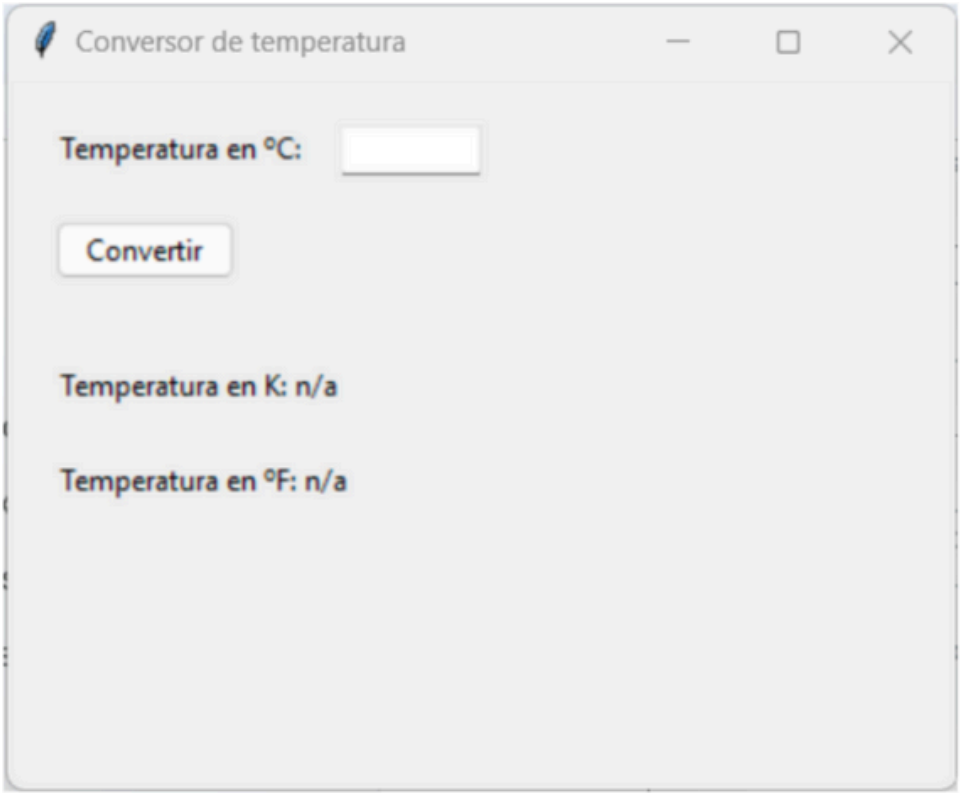


Como se observa, la lógica es siempre la misma: primero la creación del control, luego su posicionamiento. La clase Entry() no lleva argumento text puesto que se espera que el usuario ingrese allí un dato.

Respecto del diseño de la interfaz nos resta únicamente agregar las dos etiquetas en las cuales se mostrarán los resultados de la conversión, o sea, los valores en Kelvin y grados Fahrenheit. vamos a agregarlas:

```
import tkinter as tk
from tkinter import ttk

ventana = tk.Tk()
ventana.title("Conversor de temperatura")
ventana.config(width=400, height=300)

etiqueta_temp_celsius = ttk.Label(text="Temperatura en oC:")
etiqueta_temp_celsius.place(x=20, y=20)

caja_temp_celsius = ttk.Entry()
caja_temp_celsius.place(x=140, y=20, width=60)

boton_convertir = ttk.Button(text="Convertir")
boton_convertir.place(x=20, y=60)

etiqueta_temp_kelvin = ttk.Label(text="Temperatura en K: n/a")
etiqueta_temp_kelvin.place(x=20, y=120)

etiqueta_temp_fahrenheit = ttk.Label(text="Temperatura en oF: n/a")
etiqueta_temp_fahrenheit.place(x=20, y=160)

ventana.mainloop()
```

Y se vería de esta forma:

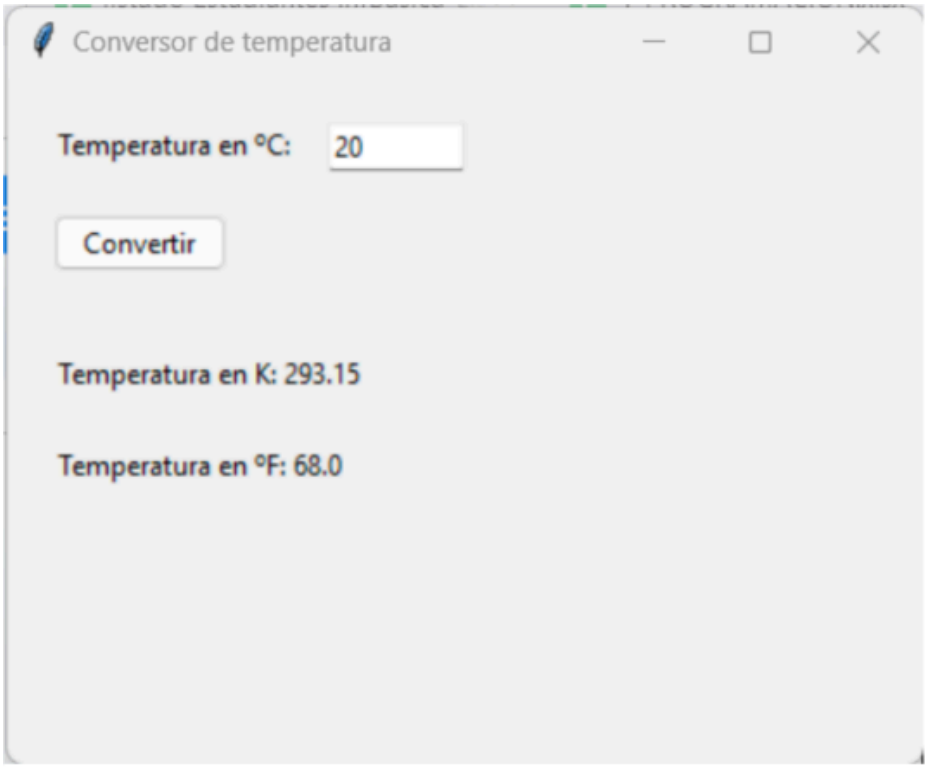


Hasta aquí el diseño de la ventana. Ahora debemos hacer que efectivamente al presionar el botón nuestro programa convierta la temperatura ingresada en la caja de texto. Para ello debemos crear una función llamada convertir_temp() y asociarla al boton_convertir vía el argumento command. Luego, desde el interior de la función obtenemos el contenido de la caja de texto vía el método get() y arrojamos el resultado de la conversión en las etiqueta_temp_kelvin y etiqueta_temp_fahrenheit vía el método config().

```
import tkinter as tk
from tkinter import ttk
```

```
def convertir_temp():
  temp_celsius = float(caja_temp_celsius.get())
  temp_kelvin = temp_celsius + 273.15
  temp_fahrenheit = temp_celsius*1.8 + 32
  etiqueta_temp_kelvin.config(text=f"Temperatura en K: {temp_kelvin}")
  etiqueta_temp_fahrenheit.config(
      text=f"Temperatura en oF: {temp_fahrenheit}")

ventana = tk.Tk()
ventana.title("Conversor de temperatura")
ventana.config(width=400, height=300)

etiqueta_temp_celsius = ttk.Label(text="Temperatura en oC:")
etiqueta_temp_celsius.place(x=20, y=20)

caja_temp_celsius = ttk.Entry()
caja_temp_celsius.place(x=140, y=20, width=60)

boton_convertir = ttk.Button(text="Convertir",
command=convertir_temp)
boton_convertir.place(x=20, y=60)

etiqueta_temp_kelvin = ttk.Label(text="Temperatura en K: n/a")
etiqueta_temp_kelvin.place(x=20, y=120)

etiqueta_temp_fahrenheit = ttk.Label(text="Temperatura en oF:
n/a")
etiqueta_temp_fahrenheit.place(x=20, y=160)

ventana.mainloop()
```

Y al ingresar un valor, tendríamos algo así:



La función convertir_temp() será invocada por Tk cada vez que el usuario presione el boton_convertir. El contenido de la caja de texto hemos tenido que convertirlo a un número de coma flotante vía la función incorporada float(), ya que por defecto el resultado de get() es siempre una cadena, similar a cuando trabajamos con input()

Hemos creado el aplicativo de conversor de temperatura, lo hemos creado de forma tradicional, con un enfoque procedimental, ahora cómo se vería este mismo aplicativo creado con POO.

Enfoque POO

El enfoque orientado a objetos empaqueta toda nuestra aplicación dentro de una clase, en este caso de manera genérica le hemos puesto Aplicación, sin embargo, el nombre de la clase puede ser algo descriptivo y enfocado a sus características o el molde que proporciona.

La clase Recibe la clase Frame, del paquete ttk, que es un widget agrupador de otros widgets.

Luego inicializa el constructor con self, que hace referencia a sí mismo, es decir a la propia clase, y con parent que es el identificador del elemento primario o la cadena vacía para crear un nuevo elemento de nivel superior.

Ahora con super, heredamos el método constructor de la clase superior e iniciamos la definición y colocación de los controles o widgets.

self.nombre_elemento hace referencia a cada uno de los atributos o variables creadas para nuestra clase que empata con los widgets del paquete y la siguiente línea utiliza place() para ubicarlo de manera absoluta en la ventana

```
import tkinter as tk
from tkinter import ttk

class Aplicacion(ttk.Frame):
  def __init__(self, parent):
      super().__init__(parent)

      self.etiqueta_temp_celsius = ttk.Label(parent, text=
      "Temperatura en oC:")
      self.etiqueta_temp_celsius.place(x=20, y=20)

      self.caja_temp_celsius = ttk.Entry(parent)
      self.caja_temp_celsius.place(x=140, y=20, width=60)

      self.boton_convertir = ttk.Button(parent, text="Convertir",
      command=self.convertir_temp)
      self.boton_convertir.place(x=20, y=60)

      self.etiqueta_temp_kelvin = ttk.Label(parent, text=
      "Temperatura en K: n/a")
      self.etiqueta_temp_kelvin.place(x=20, y=120)

      self.etiqueta_temp_fahrenheit = ttk.Label(parent, text=
      "Temperatura en oF:n/a")
      self.etiqueta_temp_fahrenheit.place(x=20, y=160)

def convertir_temp(self):
```

```
        temp_celsius = float(self.caja_temp_celsius.get())

        temp_kelvin = temp_celsius + 273.15
        temp_fahrenheit = temp_celsius*1.8 + 32

        self.etiqueta_temp_kelvin.config(text=f"Temperatura en K:
        {temp_kelvin}")
        self.etiqueta_temp_fahrenheit.config(text=f"Temperatura en
        oF:{temp_fahrenheit}")

ventana = tk.Tk()
ventana.title("Conversor de temperatura")
ventana.config(width=400, height=300)

app = Aplicacion(ventana)

ventana.mainloop()
```

Finalmente encontramos el botón que cuenta con el método command en donde se llama a self.convetir_temp, haciendo referencia a un método llamado convertir_temp definido dentro de la misma clase, método que encontramos al final con la misma lógica de la versión anterior.

Es así como el mismo problema puede ser abordado desde dos perspectivas diferentes y obtener el mismo resultado.

Ejercitación 2

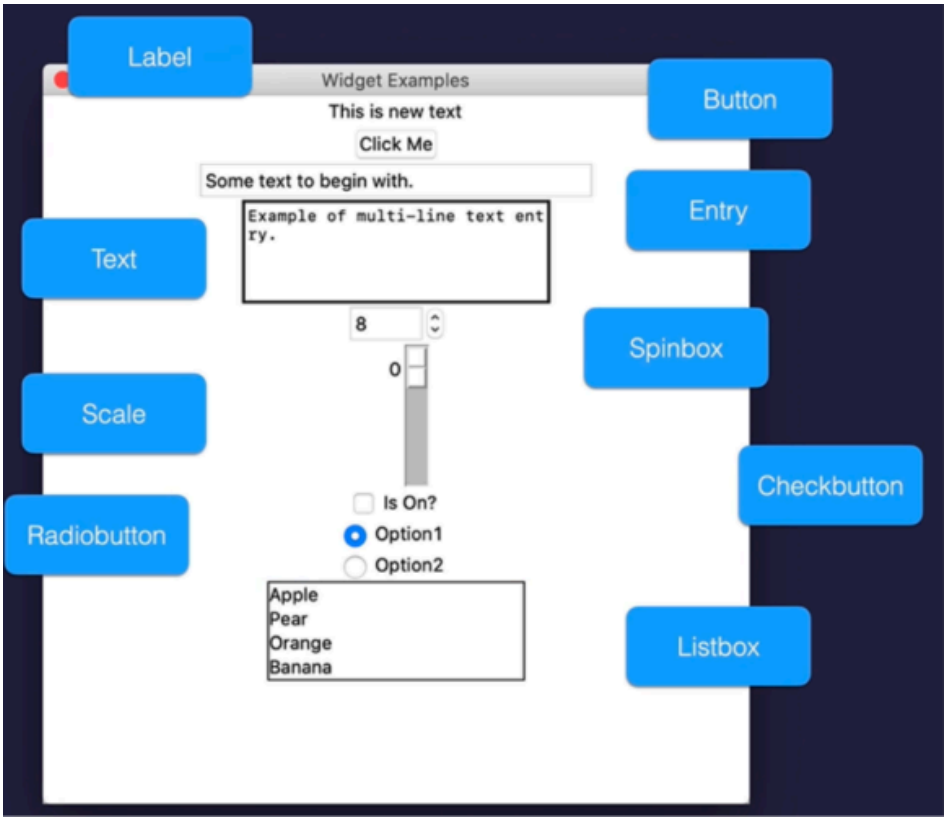
Ahora es su turno, vamos a crear el aplicativo conversor de medidas de distancia, utilizando los mismo principios, vamos a crear los controles necesarios y los métodos de conversión para mostrar toda la información asociada con sus equivalentes en medidas por ejemplo, de kilómetros a millas, metros, pies.

Widgets, Controls o Componentes

Dentro del módulo nos podemos encontrar diferentes componentes que nos ayudan a definir nuestra interfaz de usuario, a estos componentes como ya hemos visto, se les conoce como widgets, y van desde widgets contenedores generales, hasta elementos muy puntuales.

Dentro de los widgets más utilizados podemos encontrar:

- La ventana
- El botón
- Cajas de texto
- Etiquetas
- Botón de chequeo
- Menús
- Separadores



La documentación de la herramienta nos entrega un listado de los widgets disponibles y sus propiedades a la hora de crearlos.

Tk Commands, version 8.6.14

Aquí presentamos algunos de ellos

```
import tkinter as tk
from tkinter import ttk

window = tk.Tk()
window.title("Widgets with TKINTER")
window.geometry("400x800")
#window.config(width=400, height=400)
#window.minsize(width=400, height=400)

my_label = ttk.Label(window, text="Hello World!", font=("Arial", 24))
my_label2 = ttk.Label(window, text="Hello World!", font=("Arial", 24))
# En una posición específica
# my_label.place(x=20, y=20)
# En el centro de la ventana

my_label.pack(pady=10)
my_label2.pack(pady=20)#my_label.pack(pady=20, side="bottom")
#my_label.config(foreground="red", background="black")
my_label.config(anchor="center")

my_label["text"] = "New Text"
my_label.config(text="New Text 2")
```

```
# Buttons
def button_clicked():
  print("Button was clicked!")
  my_label.config(text="Button was clicked!")
  my_label2.config(text=input.get())

# button = ttk.Button(window, text="Click Me!")
button = ttk.Button(window, text="Click Me!",
command=button_clicked)
button.pack(pady=10)
# Entry
input = ttk.Entry(window, width=30)
input.pack(pady=10)#Text
text = tk.Text(height=5, width=30)
#Puts cursor in textbox.
text.focus()
#Adds some text to begin with.
# text.insert(END,"Example of multi-line text entry.")
text.insert("1.0","Example of multi-line text entry.")
#Get's current value in textbox at line 1, character 0
print(text.get("1.0","end"))
text.pack(pady=10)#Spinbox
def spinbox_used():
  #gets the current value in spinbox.
  print(spinbox.get())
spinbox = ttk.Spinbox(from_=0, to=10, width=5,
command=spinbox_used)
spinbox.pack(pady=10)#Scale
#Called with current scale value.
def scale_used(value):
  print(value)
```

```
scale = ttk.Scale(from_=0, to=100, command=scale_used)
scale.pack(pady=10)#Checkbutton
def checkbutton_used():
  #Prints 1 if On button checked, otherwise 0.
print(checked_state.get())
#variable to hold on to checked state, 0 is off, 1 is on.
checked_state = tk.BooleanVar()
# checked_state = IntVar()
checkbutton = ttk.Checkbutton(text="Is On?",
variable=checked_state,
command=checkbutton_used)
checked_state.get()
checkbutton.pack(pady=10)#Radiobutton
def radio_used():
  print(radio_state.get())
#Variable to hold on to which radio button value is checked.
radio_state = tk.IntVar()
radiobutton1 = ttk.Radiobutton(text="Option1", value=1,
variable=radio_state,
command=radio_used)
radiobutton2 = ttk.Radiobutton(text="Option2", value=2,
variable=radio_state,
command=radio_used)
radiobutton1.pack(pady=10)
radiobutton2.pack(pady=10)
#Listbox
def listbox_used(event):
  # Gets current selection from listbox
  print(listbox.get(listbox.curselection()))
```

```
listbox = tk.Listbox(height=4)
fruits = ["Apple","Pear","Orange","Banana"]
for item in fruits:
listbox.insert(fruits.index(item), item)
listbox.bind("<<ListboxSelect>>", listbox_used)
listbox.pack(pady=10)

window.mainloop()
```

Ejercitación calculadora

Ahora vamos a realizar la ejercitación para el desarrollo de una calculadora sencilla, la idea es ver un manejo adicional de la ubicación de widgets en pantalla y es llamado grid(), ya hemos visto place para valores absolutos y pack para ubicar por defecto en el centro o en algún otro punto según modifiquemos sus valores.

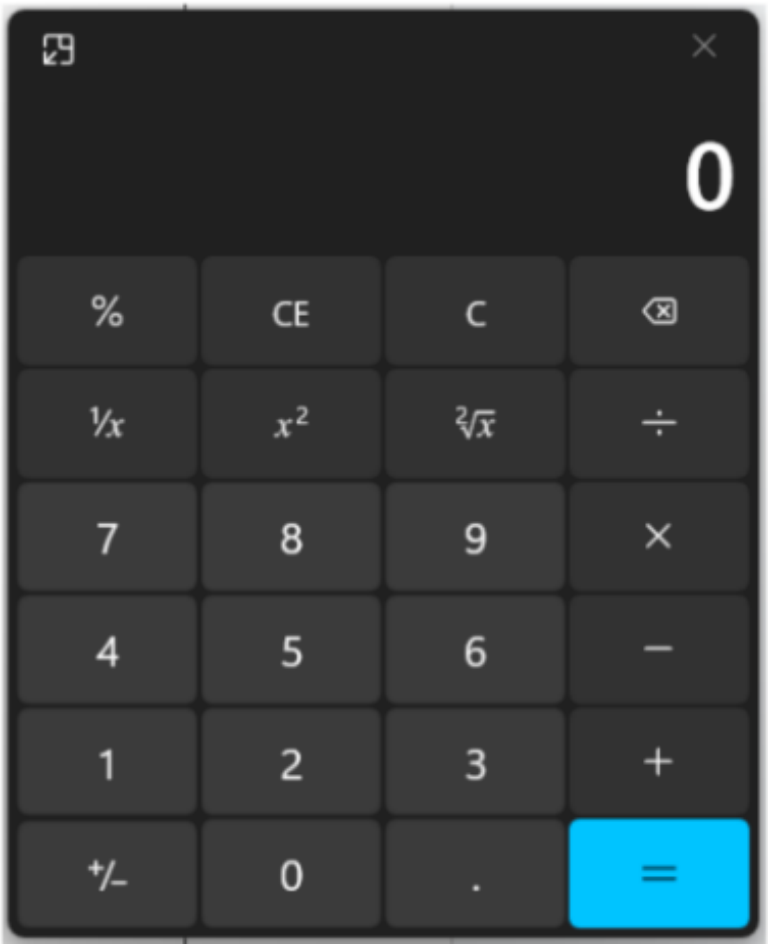
Lo primero es ver un poco de la ubicación en rejilla o grid, imagina que tu ventana o programa es una hoja cuadriculada, que puede tener tantas filas y columnas como necesites, las filas son horizontales, y las columnas verticales

Fila 0 / columna 0	Fila 0 / Columna 1	Fila 0 / Columna 2
Fila 1 / columna 0	Fila 1 / Columna 1	Fila 1 / Columna 2
Fila 2 / columna 0	Fila 2 / Columna 1	Fila 2 / Columna 2

Entonces si queremos definir un elemento ubicado en alguna posición, solo necesitamos indicar la fila y la columna:

```
button.grid(row=0, column=0)
```

Ahora revisemos el diseño de una calculadora, como por ejemplo la que viene con windows, si revisamos su diseño podemos trazar las líneas que componen el diseño de rejilla.



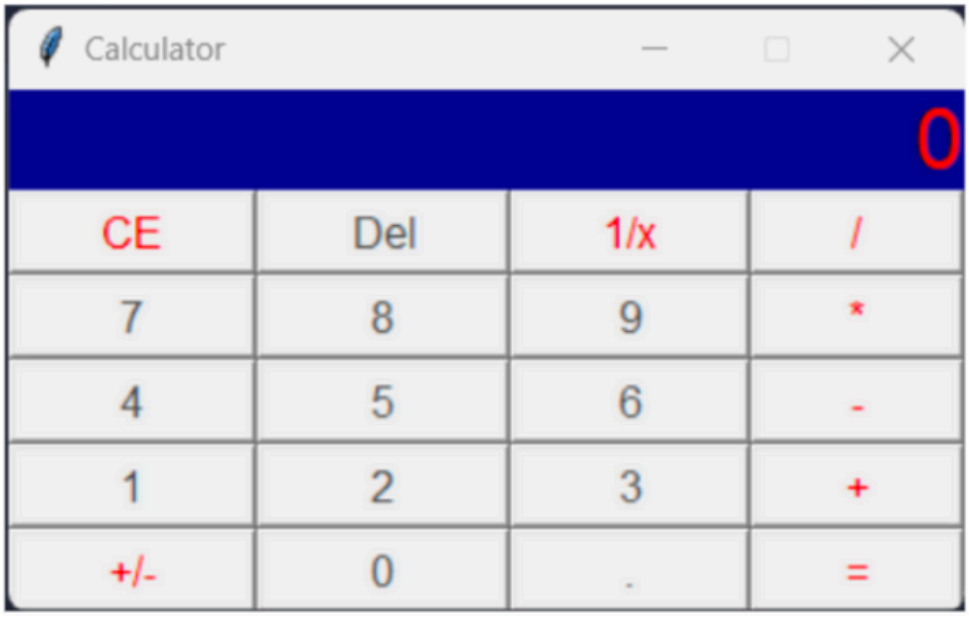
Junto con esto podemos determinar los elementos o widgets que necesitamos, como podemos ver utilizamos un montón de botones para los números y las operaciones y un espacio en el que podemos escribir utilizando el teclado y también donde se visualizan los datos al dar clic sobre los botones.

El diseño de rejilla nos sugiere que necesitamos al menos 4 columnas y unas 5 filas, considerando solo operaciones básicas y los botones para limpiar/borrar.

El diseño de la calculadora a construir entonces puede definirse así:

			80085
CE	DEL	1/x	/
7	8	9	*
4	5	6	-
1	2	3	+
+/-	0	.	=

La solución inicial a la calculadora utilizando POO se vería así:



Y su código es el siguiente:

```
from tkinter import *
from tkinter import messagebox

class Pycalc(Frame):

  def __init__(self, master,*args,**kwargs):
     Frame.__init__(self, master,*args,**kwargs)
     self.parent = master
     self.grid()
     self.createWidgets()

  def deleteLastCharacter(self):
     textLength = len(self.display.get())

     if textLength >= 1:
       self.display.delete(textLength - 1, END)
     if textLength == 1:
       self.replaceText("0")

  def replaceText(self, text):
     self.display.delete(0, END)
     self.display.insert(0, text)

  def append(self, text):
     actualText = self.display.get()
     textLength = len(actualText)
     if actualText =="0":
       self.replaceText(text)
     else:
       self.display.insert(textLength, text)
```

```
def evaluate(self):
    try:
        self.replaceText(eval(self.display.get()))
    except (SyntaxError, AttributeError):
        messagebox.showerror("Error","Syntax Error")
        self.replaceText("0")
    except ZeroDivisionError:
        messagebox.showerror("Error","Cannot Divide by 0")
        self.replaceText("0")

def containsSigns(self):
    operatorList = ["*","/","+","-"]
    display = self.display.get()
    for c in display:
        if c in operatorList:
            return True
    return False

def changeSign(self):
    if self.containsSigns():
        self.evaluate()
    firstChar = self.display.get()[0]
    if firstChar =="0":
        pass
    elif firstChar =="-":
        self.display.delete(0)
    else:
        self.display.insert(0,"-")

def inverse(self):
    self.display.insert(0,"1/(")
    self.append(")")
    self.evaluate()
```

```
  def createWidgets(self):
        self.display = Entry(self, font=("Arial", 24), relief=RAISED,
        justify=RIGHT,
bg='darkblue', fg='red', borderwidth=0)
    self.display.insert(0,"0")
    self.display.grid(row=0, column=0, columnspan=4, sticky="nsew")

    self.ceButton = Button(self, font=("Arial", 12), fg='red', text="CE",
highlightbackground='red', command=lambda: self.replaceText("0"))
    self.ceButton.grid(row=1, column=0, sticky="nsew")
        self.inverseButton = Button(self, font=("Arial", 12), fg='red',
        text="1/x",
highlightbackground='lightgrey', command=lambda: self.inverse())
    self.inverseButton.grid(row=1, column=2, sticky="nsew")
        self.delButton = Button(self, font=("Arial", 12), fg='#4d4d4d',
        text="Del",
highlightbackground='red',                    command=lambda:
self.deleteLastCharacter())
    self.delButton.grid(row=1, column=1, sticky="nsew")
            self.divButton = Button(self, font=("Arial", 12), fg='red', text=
            "/",
highlightbackground='lightgrey', command=lambda: self.append("/"))
    self.divButton.grid(row=1, column=3, sticky="nsew")
self.sevenButton = Button(self, font=("Arial", 12), fg='#4d4d4d', text=
"7",
highlightbackground='black', command=lambda: self.append("7"))
    self.sevenButton.grid(row=2, column=0, sticky="nsew")
        self.eightButton = Button(self, font=("Arial", 12), fg='#4d4d4d',
        text="8",
highlightbackground='black', command=lambda: self.append("8"))
    self.eightButton.grid(row=2, column=1, sticky="nsew")
        self.nineButton = Button(self, font=("Arial", 12), fg='#4d4d4d',
        text="9"
```

```
highlightbackground='black', command=lambda: self.append("9"))
    self.nineButton.grid(row=2, column=2, sticky="nsew")
        self.multButton = Button(self, font=("Arial", 12), fg='red',
        text="*",
highlightbackground='lightgrey',                    command=lambda:
self.append("*"))
    self.multButton.grid(row=2, column=3, sticky="nsew")

    self.fourButton = Button(self, font=("Arial", 12), fg='#4d4d4d',
    text="4",
highlightbackground='black', command=lambda: self.append("4"))
    self.fourButton.grid(row=3, column=0, sticky="nsew")
        self.fiveButton = Button(self, font=("Arial", 12), fg='#4d4d4d',
        text="5",
highlightbackground='black', command=lambda: self.append("5"))
    self.fiveButton.grid(row=3, column=1, sticky="nsew")
        self.sixButton = Button(self, font=("Arial", 12), fg='#4d4d4d',
        text="6",
highlightbackground='black', command=lambda: self.append("6"))
    self.sixButton.grid(row=3, column=2, sticky="nsew")
        self.minusButton = Button(self, font=("Arial", 12), fg='red',
        text="-",
highlightbackground='lightgrey',  command=lambda:  self.append("-
"))
    self.minusButton.grid(row=3, column=3, sticky="nsew")
```

```
        self.oneButton = Button(self, font=("Arial", 12), fg='#4d4d4d',
        text="1",
highlightbackground='black', command=lambda: self.append("1"))
    self.oneButton.grid(row=4, column=0, sticky="nsew")
        self.twoButton = Button(self, font=("Arial", 12), fg='#4d4d4d',
        text="2",
highlightbackground='black', command=lambda: self.append("2"))
    self.twoButton.grid(row=4, column=1, sticky="nsew")
      self.threeButton = Button(self, font=("Arial", 12), fg='#4d4d4d',
      text="3",
highlightbackground='black', command=lambda: self.append("3"))
    self.threeButton.grid(row=4, column=2, sticky="nsew")

        self.plusButton = Button(self, font=("Arial", 12), fg='red',
        text="+",
highlightbackground='lightgrey', command=lambda: self.append("+"))
    self.plusButton.grid(row=4, column=3, sticky="nsew")
        self.negToggleButton = Button(self, font=("Arial", 12), fg='red',
        text="+/-",
highlightbackground='lightgrey',                command=lambda:
self.changeSign())
    self.negToggleButton.grid(row=5, column=0, sticky="nsew")
        self.zeroButton = Button(self, font=("Arial", 12), fg='#4d4d4d',
        text="0",
highlightbackground='black', command=lambda: self.append("0"))
    self.zeroButton.grid(row=5, column=1, sticky="nsew")
      self.decimalButton = Button(self, font=("Arial", 12), fg='#4d4d4d',
      text=".",
highlightbackground='lightgrey', command=lambda:  self.append("."))
    self.decimalButton.grid(row=5, column=2, sticky="nsew")
        self.equalsButton = Button(self, font=("Arial", 12), fg='red',
        text="=",
```

```
highlightbackground='lightgrey', command=lambda: self.evaluate())
    self.equalsButton.grid(row=5, column=3, sticky="nsew")

Calculator = Tk()
Calculator.title("Calculator")
Calculator.resizable(False, False)
Calculator.config(cursor="target")
root = Pycalc(Calculator).grid()
Calculator.mainloop()
```

El reto consiste en modificar el código de la calculadora para hacerla de la forma no orientada a objetos, conservando sus funcionalidades y cambiar su apariencia para utilizar (en donde sea posible), componentes o controles temáticos, utilizando ttk.

La recomendación es construir primero todo el layout de la calculadora para luego agregar las funcionalidades, algunas propiedades que se usan en el ejemplo entregado, puede ser que no se definen de la misma forma utilizando la forma temática o clásica.

Tenga en cuenta los problemas de la división por 0 y que los únicos valores aceptados sean numéricos, ya sean enteros o flotantes.