Práctica Interfaces Gráficas: Calculadora

Para programar la calculadora hice una clase llamada “calculadora”, esta recibe como argumento la ventana o frame donde se mostrará y tiene como función de iniciación la creación de la interfaz gráfica. El definirla en una clase permitirá que se pueda reutilizar el código con mayor facilidad en el futuro.

La ejecución de la calculadora se da en las últimas líneas del archivo en las que se crea la ventana principal y se pasa como argumento a la clase calculadora que está definida justo arriba.

```
window = Tk()
window.config(bg="black")
window.resizable(False, False)
myFont = font.Font(family='Helvetica')
window.title('Calculadora')
window.iconbitmap('calculator.ico')
mywin = calculadora(window)
window.mainloop()
```

El código mostrado en la imagen anterior es son las líneas que se ejecutan cada vez que ejecutamos el archivo, el código mostrado en las siguientes secciones es parte de la clase “calculadora”.

Variables iniciales

Antes de crear los botones de la interfaz, se declaran valores iniciales necesarios para ciertas funciones de la calculadora y los colores usados para los botones.

```
# COLORES DE BOTONES Y COLOR DE LETRAS
color_nums = "#595959"
color_fuente = "#e8e8e8"
color_ops = "#4d4d4d"

# VALORES INICIALES
# ÚLTIMA RESPUESTA
self.last_ans = ""
# VARIABLES MEMORIA X, Y
self.xmem = ""
self.ymem = ""
# SE PUEDE GUARDAR EN X, Y
self.savable = False
```

Display

En la parte superior de la ventana de la calculadora se muestra el display, en este se almacena lo que el usuario ingrese por medio de los botones.

```
# DISPLAY
self.display = Label(win, text="", bg="black", fg="white", font=myFont, height=2, width=1, anchor="e")
self.display.grid(column=0, row=0, columnspan=4, sticky="we")
```

Botones numéricos

Los botones numéricos se acomodan con la función “grid” y simplemente agregaran el número en cuestión al display.

```
# BOTONES DE NÚMEROS
self.b0 = Button(win, text='0', width=3, bg=color_nums, fg=color_fuente, font=myFont, command = lambda: self.addDisplay("0"))
self.b1 = Button(win, text='1', width=3, bg=color_nums, fg=color_fuente, font=myFont, command = lambda: self.addDisplay("1"))
self.b2 = Button(win, text='2', width=3, bg=color_nums, fg=color_fuente, font=myFont, command = lambda: self.addDisplay("2"))
self.b3 = Button(win, text='3', width=3, bg=color_nums, fg=color_fuente, font=myFont, command = lambda: self.addDisplay("3"))
self.b4 = Button(win, text='4', width=3, bg=color_nums, fg=color_fuente, font=myFont, command = lambda: self.addDisplay("4"))
self.b5 = Button(win, text='5', width=3, bg=color_nums, fg=color_fuente, font=myFont, command = lambda: self.addDisplay("5"))
self.b6 = Button(win, text='6', width=3, bg=color_nums, fg=color_fuente, font=myFont, command = lambda: self.addDisplay("6"))
self.b7 = Button(win, text='7', width=3, bg=color_nums, fg=color_fuente, font=myFont, command = lambda: self.addDisplay("7"))
self.b8 = Button(win, text='8', width=3, bg=color_nums, fg=color_fuente, font=myFont, command = lambda: self.addDisplay("8"))
self.b9 = Button(win, text='9', width=3, bg=color_nums, fg=color_fuente, font=myFont, command = lambda: self.addDisplay("9"))
```

```
self.b0.grid(column=0, row=6)
self.b1.grid(column=0, row=5)
self.b2.grid(column=1, row=5)
self.b3.grid(column=2, row=5)
self.b4.grid(column=0, row=4)
self.b5.grid(column=1, row=4)
self.b6.grid(column=2, row=4)
self.b7.grid(column=0, row=3)
self.b8.grid(column=1, row=3)
self.b9.grid(column=2, row=3)
```

Todos los botones que agregan algún carácter al display hacen uso de la siguiente función (que es un método de la clase “calculadora”).

```
# AGREGAR CARACTERES AL DISPLAY
def addDisplay(self, character):
    self.display["text"] += character
    self.savable = False
```

Botones de operaciones

La calculadora tiene las operaciones básicas suma, resta, multiplicación, división y potenciación, también admite el uso de paréntesis que son respetados por la jerarquía de operaciones y el uso del punto para calcular con números decimales.

```
#BOTONES DE OPERADORES
self.bdiv = Button(win, text='÷', width=3, bg=color_ops, fg=color_fuente, font=myFont, command = lambda: self.addDisplay(" ÷ "))
self.bpor = Button(win, text='×', width=3, bg=color_ops, fg=color_fuente, font=myFont, command = lambda: self.addDisplay(" × "))
self.bmas = Button(win, text='-', width=3, bg=color_ops, fg=color_fuente, font=myFont, command = lambda: self.addDisplay(" - "))
self.bmenos = Button(win, text='+', width=3, bg=color_ops, fg=color_fuente, font=myFont, command = lambda: self.addDisplay(" + "))
self.bpunto = Button(win, text='.', width=3, bg=color_ops, fg=color_fuente, font=myFont, command = lambda: self.addDisplay("."))
self.bpot = Button(win, text='^', width=3, bg=color_ops, fg=color_fuente, font=myFont, command = lambda: self.addDisplay(" ^ "))
self.bpari = Button(win, text='(', width=3, bg=color_ops, fg=color_fuente, font=myFont, command = lambda: self.addDisplay("("))
self.bpard = Button(win, text=')', width=3, bg=color_ops, fg=color_fuente, font=myFont, command = lambda: self.addDisplay(")"))
```

```
self.bdiv.grid(column=3, row=2)
self.bpor.grid(column=3, row=3)
self.bmas.grid(column=3, row=4)
self.bmenos.grid(column=3, row=5)
self.bpunto.grid(column=1, row=6)
self.bpot.grid(column=2, row=2)
self.bpari.grid(column=0, row=2)
self.bpard.grid(column=1, row=2)
```

Estos botones tienen como función agregar caracteres con el método antes mencionado, la función usada para la resolución del display será explicado más adelante.

Funciones adicionales

Como funciones adicionales están las variables de memoria x e y, que permiten guardar resultados y usarlos en cálculos posteriores; el botón "ANS", que guarda el último resultado que se haya obtenido; y por último, los botones de retroceso y borrado de display.

```
# OTROS BOTONES
self.bxmem = Button(win, text='x', width=3, bg=color_ops, fg=color_fuente, font=myFont, command = self.use_xmem)
self.bymem = Button(win, text='y', width=3, bg=color_ops, fg=color_fuente, font=myFont, command = self.use_ymem)
self.bborrar = Button(win, text='⌫', width=3, bg=color_ops, fg=color_fuente, font=myFont, command = self.backspace)
self.bans = Button(win, text='ANS', width=3, bg=color_ops, fg=color_fuente, font=(myFont,12), command = lambda: self.addDisplay(self.last_ans))
self.bdel = Button(win, text='C', width=3, bg=color_ops, fg=color_fuente, font=myFont, command = self.deleteDisplay)
```

```
self.bxmem.grid(column=0, row=1)
self.bymem.grid(column=1, row=1)
self.bborrar.grid(column=3, row=1)
self.bans.grid(column=2, row=6, sticky = "nswe")
self.bdel.grid(column=2, row=1)
```

Los botones de variables de memoria solo guardan valores justo después de haber presionado el botón igual, guardará el valor calculado en el botón de la variable de memoria que se presione, en cualquier otra situación, el botón agregará lo que tenga guardado al display.

```
def use_xmem(self):
    if(self.savable):
        self.xmem = self.display["text"]
    else:
        self.addDisplay(self.xmem)

def use_ymem(self):
    if(self.savable):
        self.ymem = self.display["text"]
    else:
        self.addDisplay(self.ymem)
```

La función "ANS" funciona de forma más sencilla, simplemente guardará el último resultado tras haber presionado igual.

La función de retroceso tiene en cuenta que los signos se insertan con un espacio antes y después de estos. El botón de borrado de display simplemente modifica el texto de la label display a una cadena de caracteres vacía.

```
# BORRA DISPLAY CARACTER POR CARACTER
def backspace(self):
    if(self.display["text"][-1:] != " "):
        self.display["text"] = self.display["text"][:-1]
    else:
        self.display["text"] = self.display["text"][:-3]
    self.savable = False
```

```
# BORRAR DISPLAY COMPLETO
def deleteDisplay(self):
    self.display["text"] = ""
    self.savable = False
```

Botón igual

El botón igual toma la expresión ingresada en el display y la resuelve usando el método "resolve", que respeta la jerarquía de operaciones y es usado recursivamente para resolver los paréntesis que haya en la expresión. Este método tiene un ciclo para cada etapa de resolución de operaciones siguiendo la jerarquía en el siguiente orden: paréntesis, potenciación, multiplicación y división, y finalmente, suma y resta.

```
# RESOLUCIÓN DE PARÉNTESIS
end_loop = False
while(not end_loop):
    end_i_loop = False
    for i in range(len(dis_str)):
        if(dis_str[i] == "("):
            count_pars = 1
            for j in range(i+1,len(dis_str)):
                if(dis_str[j] == "("):
                    count_pars += 1
                elif(dis_str[j] == ")"):
                    count_pars += -1
                if(count_pars == 0):
                    par_sol = self.resolve(dis_str[i+1:j])
                    if(dis_str[1] == "-" or dis_str[1] == "+"):
                        dis_str = "0" + dis_str
                    dis_str = dis_str[:i] + par_sol + dis_str[j+1:]
                    end_i_loop = True
                    break
            if(count_pars != 0):
                end_loop = True
                end_i_loop = True
        if(end_i_loop):
            break
    if(not "(" in dis_str):
        end_loop = True
```

```
# SEPARAMOS CADA COMPONENTE DE LA OPERACION POR
# LOS ESPACIOS QUE HAY ENTRE ELLOS
dis_sep = dis_str.split(" ")

# CONVERTIMOS A FLOAT AQUELLOS ELEMENTOS QUE NO SEAN UN OPERADOR
for i in range(len(dis_sep)):
    if(not dis_sep[i] in "+-x÷^"):
        dis_sep[i] = float(dis_sep[i])
```

```
# RESOLUCIÓN DE POTENCIAS
end_loop = False
while(not end_loop):
    for i in range(len(dis_sep)):
        if(dis_sep[i] == "^"):
            dis_sep[i] = dis_sep[i-1]**dis_sep[i+1]
            dis_sep.pop(i+1)
            dis_sep.pop(i-1)
            break
        if(i == len(dis_sep)-1):
            end_loop = True
```

```
# RESOLUCIÓN DE MULTIPLICACIONES Y DIVISIONES
end_loop = False
while(not end_loop):
    for i in range(len(dis_sep)):
        if(dis_sep[i] == "x"):
            dis_sep[i] = dis_sep[i-1]*dis_sep[i+1]
            dis_sep.pop(i+1)
            dis_sep.pop(i-1)
            break
        elif(dis_sep[i] == "÷"):
            dis_sep[i] = dis_sep[i-1]/dis_sep[i+1]
            dis_sep.pop(i+1)
            dis_sep.pop(i-1)
            break
        if(i == len(dis_sep)-1):
            end_loop = True
```

```
# RESOLUCIÓN DE SUMAS Y RESTAS
end_loop = False
while(not end_loop):
    for i in range(len(dis_sep)):
        if(dis_sep[i] == "+"):
            dis_sep[i] = dis_sep[i-1]+dis_sep[i+1]
            dis_sep.pop(i+1)
            dis_sep.pop(i-1)
            break
        elif(dis_sep[i] == "-"):
            dis_sep[i] = dis_sep[i-1]-dis_sep[i+1]
            dis_sep.pop(i+1)
            dis_sep.pop(i-1)
            break
        if(i == len(dis_sep)-1):
            end_loop = True
```

```
solucion = str(round(dis_sep[0],3))
# SI EL INPUT ORIGINAL DE LA FUNCIÓN NO FUE NINGUNO, GUARDAMOS EN EL DISPLAY,
# SI NO, LO REGRESAMOS COMO RETURN
if(dis_in == ""):
    self.display["text"] = solucion
    self.last_ans = solucion
    # SOLO CUANDO SE DA UN RESULTADO EL ESTADO DE "SALVABILIDAD" EN LAS VARIABLES
    # DE MEMORIA SE ACTIVA
    self.savable = True
else:
    return solucion
```

Ejemplos de ejecución