



INTRODUCCIÓN A LA PROGRAMACIÓN EN PYTHON

Instructor: Ing. Abner Saavedra

Email: ingenieroabnersaavedra@gmail.com

Teléfono: 0424-3174308

Fecha: 12-02-2024

Material: Ing. Neptalí Piña

GracoSoft Centro Empresarial Plaza Madrid, piso 9, oficinas 9-7 a la 9-10



Unidad 4:

Manejo de excepciones e interfaz gráfica con Tkinter.



Contenido

- ❖ Excepciones
- ❖ ¿Qué es tkinter?
- ❖ Empezar con tkinter
- ❖ Aplicación orientada a objetos
- ❖ Widgets
 - ✓ Label
 - ✓ Establecer tamaño de una ventana
 - ✓ Button
 - Manejar el evento click de un botón
 - ✓ Entry



- ✓ Radiobutton
- ✓ Checkbutton
- ✓ Combobox
- ✓ Dialogs (MessageBox)
 - Mostrar diálogos de pregunta y respuesta
- ❖ Configuración
- ❖ Ejercicios



Excepciones

Los errores de ejecución son llamados comúnmente *excepciones*. Durante la ejecución de un programa, si dentro de una función surge una excepción y la función no la maneja, la excepción se propaga hacia la función que la invocó, si esta otra tampoco la maneja, la excepción continua propagándose hasta llegar a la función inicial del programa y si esta tampoco la maneja se interrumpe la ejecución del programa. Veamos entonces como manejar excepciones.





Para el manejo de excepciones los lenguajes proveen ciertas palabras reservadas, que nos permiten manejar las excepciones que puedan surgir y tomar acciones de recuperación para evitar la interrupción del programa o, al menos, para realizar algunas acciones adicionales antes de interrumpir el programa.

En el caso de Python, el manejo de excepciones se hace mediante los bloques que utilizan las sentencias **try**, **except** y **finally**.

Veamos un pequeño programa que lanzaría una excepción al intentar dividir 1 entre 0.

```
6 def division(a, b):  
5     return a / b  
4 def calcular():  
3     division(1, 0)  
2 calcular()  
1
```

```
Traceback (most recent call last):  
  File "prueba.py", line 5, in <module>  
    calcular()  
  File "prueba.py", line 4, in calcular  
    division(1, 0)  
  File "prueba.py", line 2, in division  
    return a / b  
ZeroDivisionError: division by zero
```



En este caso, se levantó la excepción `ZeroDivisionError` cuando se quiso hacer la división. Para evitar que se levante la excepción y se detenga la ejecución del programa, se utiliza el bloque **try-except**.

```
15 def division(a, b):  
14     return a / b  
13  
12  
11 def calcular():  
10     division(1, 0)  
9  
8  
7 try:  
6     calcular()  
5 except Exception:  
4     print("No se permite la división por cero")  
3
```



¿Qué es tkinter?

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).

La mayor fortaleza de Tkinter es su ubicuidad y simplicidad. Funciona de forma inmediata en la mayoría de las plataformas (Linux, OSX, Windows) y se completa con una amplia gama de widgets necesarios para las tareas más comunes (botones, etiquetas, lienzos de dibujo, texto de varias líneas, etc.).



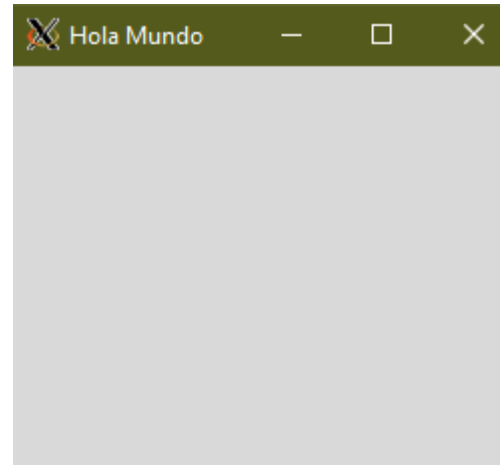
Empezar con tkinter

Problema:

Mostrar una ventana y que en su título aparezca el mensaje 'Hola Mundo'.

El programa en Python haciendo uso del módulo 'tkinter' requiere el siguiente algoritmo:

```
1 import tkinter as tk
2
3 ventana1=tk.Tk()
4 ventana1.title("Hola Mundo")
5 ventana1.mainloop()
6
```





Lo primero que hacemos es importar el módulo tkinter y le definimos un alias luego de la palabra clave **as**. Es más fácil luego escribir 'tk' en lugar de 'tkinter':

```
1 import tkinter as tk
```

La librería tkinter está codificada con la metodología de programación orientada a objetos. El módulo 'tkinter' tiene una clase llamada 'Tk' que representa una ventana. Debemos crear un objeto de dicha clase:

```
3 ventana1=tk.Tk()
```

Seguidamente llamamos al método 'title' y le pasamos un string con el mensaje que queremos que aparezca en la barra del título de la ventana:

```
4 ventana1.title("Ho la Mundo")
```

Para que se muestre la ventana en el monitor debemos llamar por último al método 'mainloop' que pertenece a la clase Tk:

```
5 ventana1.mainloop()
```



Aplicación orientada a objetos

El programa anterior modificado con POO queda:

```
1 import tkinter as tk
2
3 class Aplicacion:
4     def __init__(self):
5         self.ventana1=tk.Tk()
6         self.ventana1.title("Hola Mundo")
7         self.ventana1.mainloop()
8
9
10 def main():
11     aplicacion1=Aplicacion()
12
13
14 if __name__ == "__main__":
15     main()
16
```

Planteamos una clase llamada 'Aplicacion' y en su método '__init__' creamos el objeto de la clase 'Tk' para que se muestre la ventana.

Debemos crear luego un objeto de la clase 'Aplicacion'.



Widgets

- ❖ **Label:** utilizado para mostrar textos. Suele ser texto estático, de ahí que se llame label o **etiqueta** de texto.
- ❖ **button:** es utilizado para ejecutar una acción al ser usado.
- ❖ **Entry:** etiqueta que permite introducir texto corto (típico de formularios).
- ❖ **Radiobutton:** permite elegir una opción entre varias.
- ❖ **Checkbutton:** permite elegir varias de las opciones propuestas.
- ❖ **Combobox:** permite seleccionar un string de un conjunto de items que se despliegan.
- ❖ **Dialogs (MessageBox):** ventana emergente (o *pop-up*).



Label

Para agregar una etiqueta a nuestro ejemplo anterior, crearemos una etiqueta usando la clase label de la siguiente manera:

```
self.label= tk.Label(self.ventana1, text="Hello")
```

Luego estableceremos su posición en el formulario utilizando el método **place** con su ubicación, de esta manera:

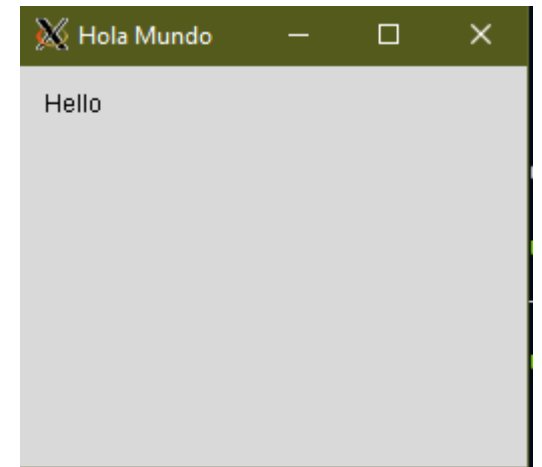
```
self.label.place(x=10,y=10)
```

Donde x,y representan un sistema de coordenadas. X representan la coordenada horizontal empezando desde la izquierda. Y y representa la coordenada vertical empezando desde arriba.



Entonces el código completo se verá de esta manera:

```
1 import tkinter as tk
2
3 class Aplicacion:
4     def __init__(self):
5         self.ventana1=tk.Tk()
6         self.ventana1.title("Hola Mundo")
7
8         self.label= tk.Label(self.ventana1, text="Hello")
9         self.label.place(x=10,y=10)
10
11         self.ventana1.mainloop()
12
13
14 def main():
15     aplicacion1=Aplicacion()
16
17
18 if __name__ == "__main__":
19     main()
20
```



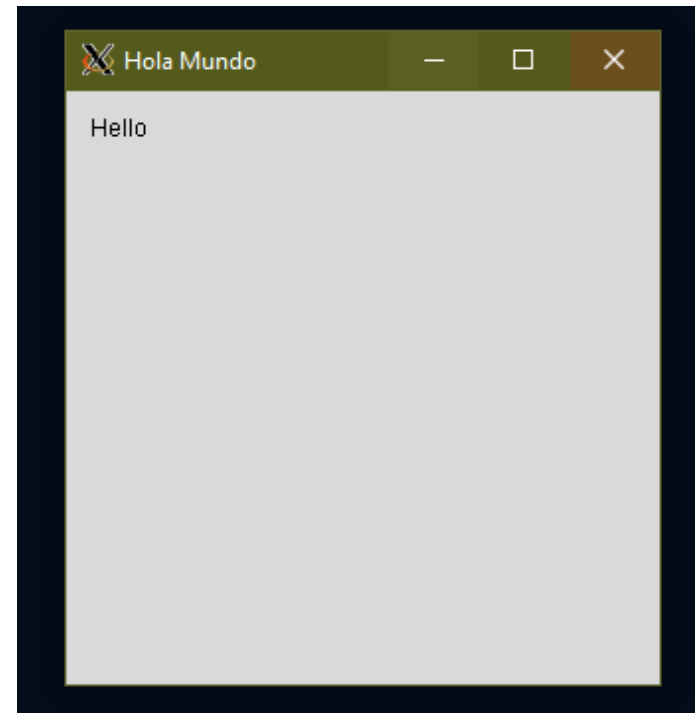


Establecer el tamaño de una ventana

Podemos establecer el tamaño predeterminado de ventana, usando la función **geometry**, de esta manera:

```
self.ventana1.geometry("300x300")
```

La línea anterior establece el ancho de la ventana en 300 píxeles y la altura en 300 píxeles.

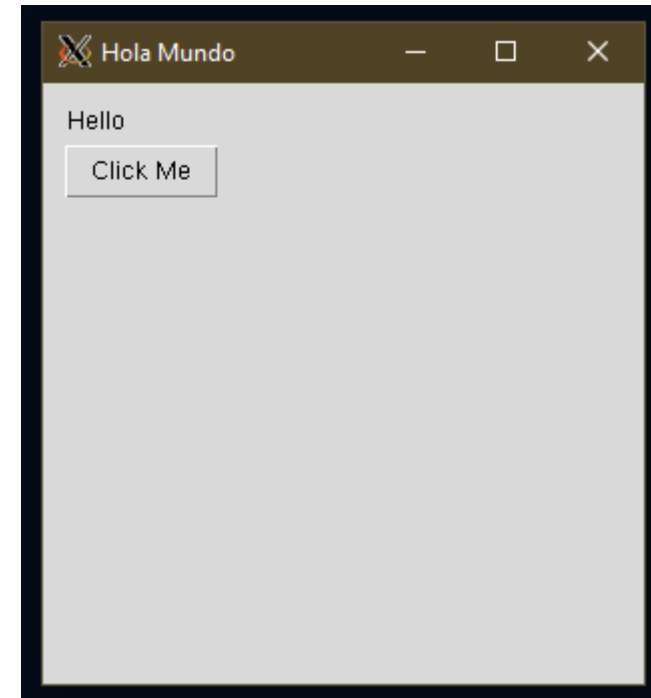




Button

Comencemos agregando un botón a la ventana. El botón se crea y se agrega a la ventana de la misma manera que la etiqueta:

```
self.btn = tk.Button(self.ventana1, text="Click Me")  
self.btn.place(x=10,y=30)
```





Manejar el evento click de un botón

Primero, escribiremos el método que necesitamos ejecutar cuando se haga click en el botón:

```
def click(self):  
    self.label.configure(text="el botón fue pulsado!!")
```

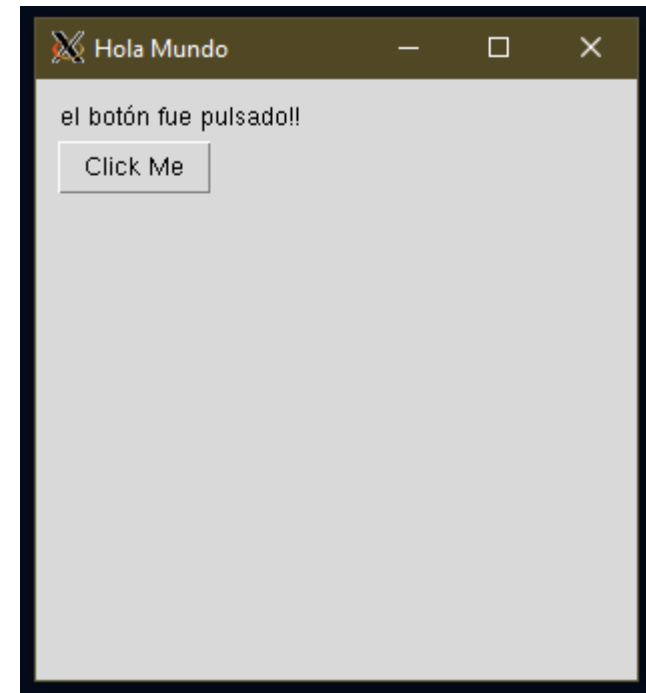
En este caso cambiara el texto del **label** usando la función **configure**

Luego le asignamos el método a el botón, especificando el argumento al parámetro **command**, de esta manera:

```
self.btn = tk.Button(self.ventana1, text="Click Me", command=self.click)
```



```
1 import tkinter as tk
2
3 class Aplicacion:
4     def __init__(self):
5         self.ventana1=tk.Tk()
6         self.ventana1.title("Hola Mundo")
7
8         # Tamaño de la ventana
9         self.ventana1.geometry("300x300")
10
11        # Label (etiqueta)
12        self.label= tk.Label(self.ventana1, text="Hello")
13        self.label.place(x=10,y=10)
14
15        # Botón
16        self.btn = tk.Button(self.ventana1, text="Click Me", command=self.click)
17        self.btn.place(x=10,y=30)
18
19
20        self.ventana1.mainloop()
21
22    def click(self):
23        self.label.configure(text="el botón fue pulsado!!")
24
25
26 def main():
27     aplicacion1=Aplicacion()
28
29
30 if __name__ == "__main__":
31     main()
32
```





Entry

Puedes crear un cuadro de texto usando la clase Tkinter Entry de esta manera:

```
self.txt = tk.Entry(self.ventana1,width=10)
self.txt.place(x=10, y=70)
```

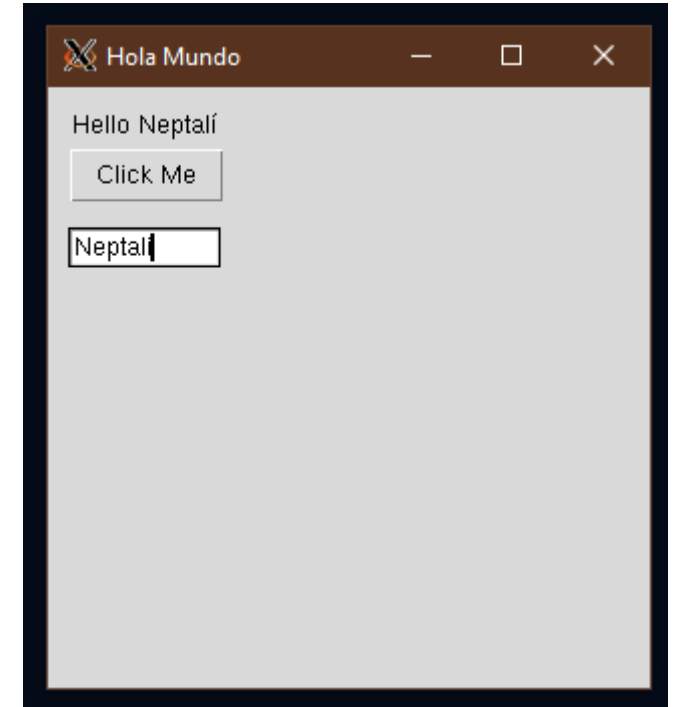
Primero, podemos obtener texto de entrada usando la función **get**. Entonces podemos escribir el código en nuestra función click, de esta manera:

```
def click(self):
    resultado = "Hello "+self.txt.get()
    self.label.configure(text=resultado)
```

Si haces click en el botón y hay un texto en el widget de entrada, se mostrará “Hello ” concatenado con el texto ingresado.



```
1 import tkinter as tk
2
3 class Aplicacion:
4     def __init__(self):
5         self.ventana1=tk.Tk()
6         self.ventana1.title("Hola Mundo")
7
8         # Tamaño de la ventana
9         self.ventana1.geometry("300x300")
10
11         # Label (etiqueta)
12         self.label= tk.Label(self.ventana1, text="Hello")
13         self.label.place(x=10,y=10)
14
15         # Botón
16         self.btn = tk.Button(self.ventana1, text="Click Me", command=self.click)
17         self.btn.place(x=10,y=30)
18
19         # Entrada de texto
20         self.txt = tk.Entry(self.ventana1,width=10)
21         self.txt.place(x=10, y=70)
22
23         self.ventana1.mainloop()
24
25     def click(self):
26         resultado = "Hello "+self.txt.get()
27         self.label.configure(text=resultado)
28
29
30
31 def main():
32     aplicacion1=Aplicacion()
33
34
35 if __name__ == "__main__":
36     main()
37
```





Radiobutton

Para agregar radio buttons, simplemente puedes usar la clase RadioButton, de esta manera:

```
self.valor = tk.IntVar()  
self.valor.set(1)  
  
self.rad1= tk.Radiobutton(self.ventana1,value=1, text="primero", variable=self.valor)  
self.rad1.place(x=10, y=90)  
  
self.rad2= tk.Radiobutton(self.ventana1, value=2, text="segundo",variable=self.valor)  
self.rad2.place(x=80, y=90)
```

Previamente hemos definido el objeto de la clase IntVar que se encuentra en el módulo tk, para luego asignarlo a las variables radioButton (rad1,rad2).

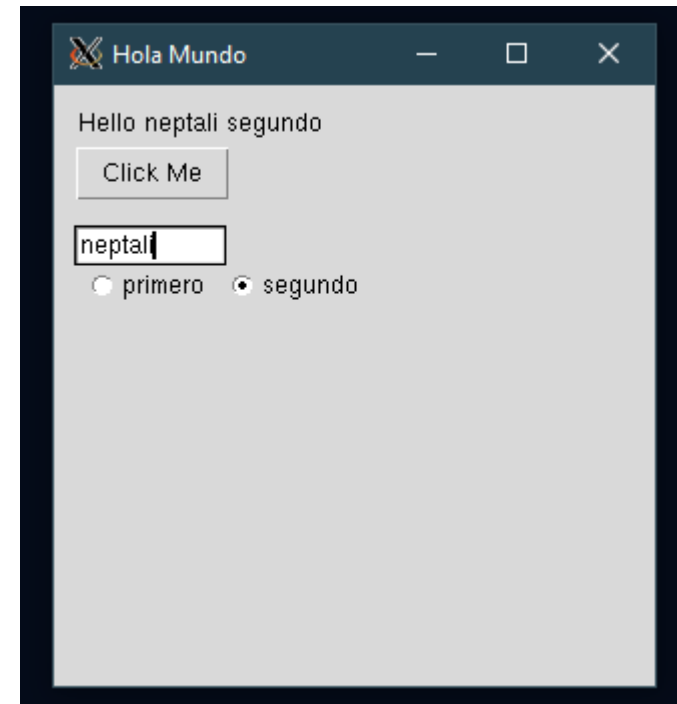
Ten en cuenta que, debes establecer el valor para cada botón de opción con un valor diferente; de lo contrario, no funcionarán.



Cambiamos la función click para mostrar la selección.

```
def click(self):  
    if self.valor.get()==1:  
        resultado = "Hello "+self.txt.get()+" primero"  
    elif self.valor.get()==2:  
        resultado = "Hello "+self.txt.get()+" segundo"  
    self.label.configure(text=resultado)
```

Ahora, concatena “primero” si el valor de la selección es 1, y concatena “segundo” si el valor de la selección es 2





Checkbox

Para crear un widget Checkbox, puedes usar la clase **Checkbox** de esta manera:

```
self.chk_estado1 = tk.BooleanVar()  
self.chk_estado1.set(True)  
self.chk_btn=tk.Checkbutton(self.ventana1, text="selección1",var=self.chk_estado1)  
self.chk_btn.place(x=10, y=120)  
  
self.chk_estado2=tk.BooleanVar()  
self.chk_btn=tk.Checkbutton(self.ventana1, text="selección2", var=self.chk_estado2)  
self.chk_btn.place(x=90, y=120)
```

Aquí creamos una variable de tipo **BooleanVar** que no es una variable Python estándar, es una variable Tkinter. Luego la pasamos a la clase **Checkbox** para establecer el estado de verificación como la línea resaltada en el ejemplo anterior.

Además, puedes usar **IntVar** en lugar de **BooleanVar** y establecer el valor en 0 o 1



Combobox

Para agregar un widget combobox, puedes usar la clase **Combobox** de la librería ttk, de esta manera:

```
2 from tkinter import ttk

dias_semana=("lunes","martes","miércoles","jueves","viernes","sábado","domingo")
self.cmb_box= ttk.Combobox(self.ventana1,values=dias_semana)
self.cmb_box.current(0)
self.cmb_box.place(x=10, y=150)
```

Como puedes ver, agregamos los elementos del comboBox usando una tupla de valores.

Para configurar el elemento seleccionado, puedes pasar el índice del elemento deseado al método `current`.

Para obtener el elemento seleccionado, puedes usar la función **get**, como se muestra a continuación:

```
self.cmb_box.get()
```




Dialogs (MessageBox)

Para mostrar un cuadro de mensaje usando Tkinter, puedes usar la librería **messagebox**, de esta manera:

```
from tkinter import messagebox  
messagebox.showinfo('Titulo del mensaje', 'Contenido del mensaje')
```

Puede mostrar un mensaje de advertencia o mensaje de error de la misma manera. Lo único que debe cambiarse es la función message.

```
messagebox.showwarning('Titulo del mensaje', 'Contenido del mensaje') #Muestr un mensaje de advertencia  
messagebox.showerror('Titulo del mensaje', 'Contenido del mensaje') #Muestra un mensaje de error
```



Mostrar diálogos de pregunta y respuesta

Para mostrar una caja de mensaje del tipo si/no al usuario, puedes usar uno de las siguientes funciones **messagebox**:

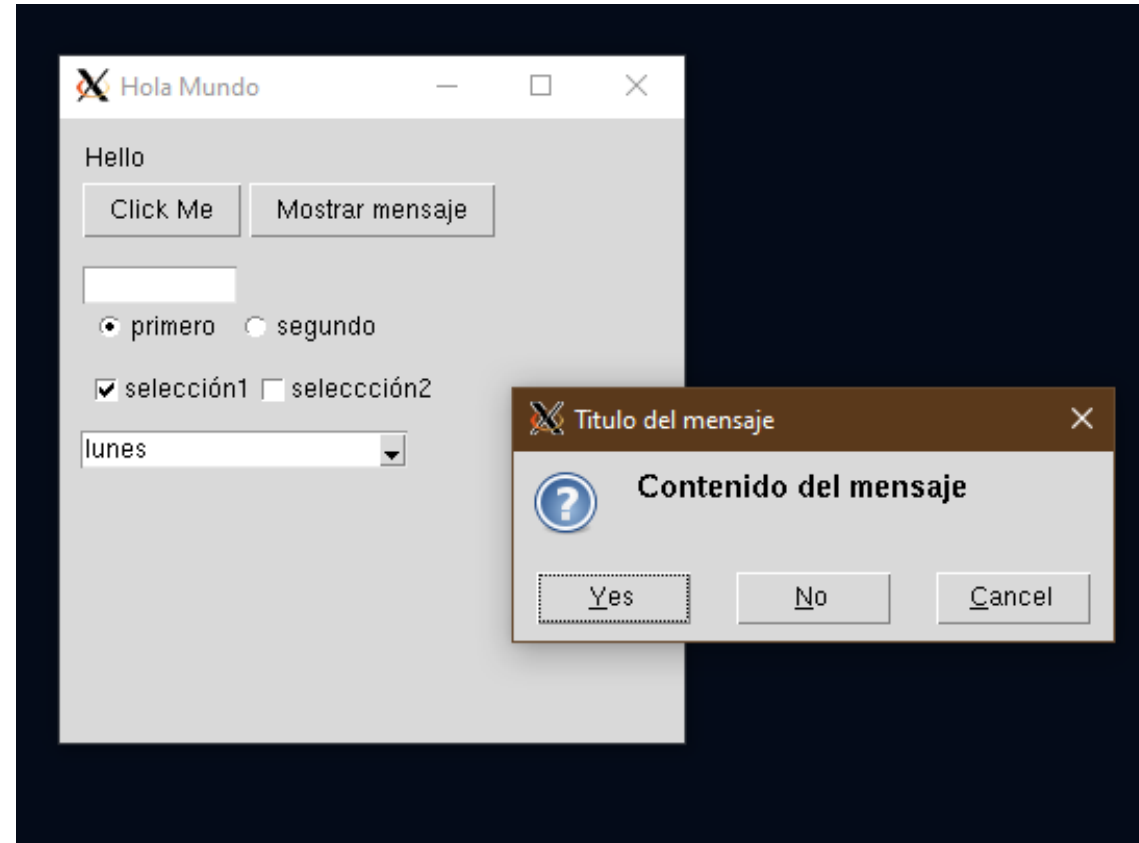
```
from tkinter import messagebox
res = messagebox.askquestion('Titulo del mensaje', 'Contenido del mensaje')
res = messagebox.asksyesno('Titulo del mensaje', 'Contenido del mensaje')
res = messagebox.asksyesnocancel('Titulo del mensaje', 'Contenido del mensaje')
res = messagebox.askokcancel('Titulo del mensaje', 'Contenido del mensaje')
res = messagebox.askretrycancel('Titulo del mensaje', 'Contenido del mensaje')
```

También puedes revisar qué botón hizo click el usuario, usando la variable **res**.

Si hace click en **OK** o **yes** o **retry**, devolverá el valor **True**; pero, si eliges **no** o **cancel**, retornará **False**.

La única función que devuelve uno de tres valores es la función **asksyesnocancel**, que

devuelve **True** o **False** o **None**.





Configuración

Para configurar un widget, simplemente llamamos a `.config()` y pasamos los argumentos que queramos modificar. Algunas opciones son:

- ❖ ***bg***: Modifica el color de fondo. Se puede indicar con el color en inglés (incluyendo modificadores, como “darkgreen”) o su código RGB en hexadecimal (“#ffffff” para blanco).
- ❖ ***fg***: Cambia el color del texto.
- ❖ ***cursor***: Modifica la forma del cursor. Algunos de los más utilizados son “gumby”, “pencil”, “watch” o “cross”.
- ❖ ***height***: Altura en líneas del componente.
- ❖ ***width***: Anchura en caracteres del componente.

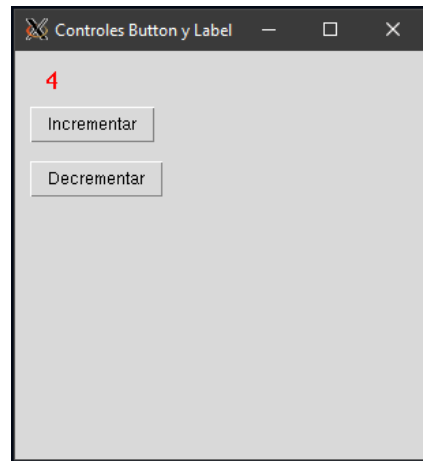


- ❖ **font:** Nos permite especificar, en una tupla con nombre de la fuente, tamaño y estilo, la fuente a utilizar en el texto del componente. Por ejemplo, `Font("Times New Roman", 24, "bold underline")`.
- ❖ **bd:** Modificamos la anchura del borde del widget.
- ❖ **relief:** Cambiamos el estilo del borde del componente. Su valor puede ser "flat", "sunken", "raised", "groove", "solid" o "ridge".
- ❖ **state:** Permite deshabilitar el componente (`state='disabled'`); por ejemplo, un *Entry* en la que no se puede escribir o un *Button* que no se puede clicar.
- ❖ **command:** De cara a que los botones realicen una acción, podemos indicar qué función ejecutar cuando se haga click en el mismo.



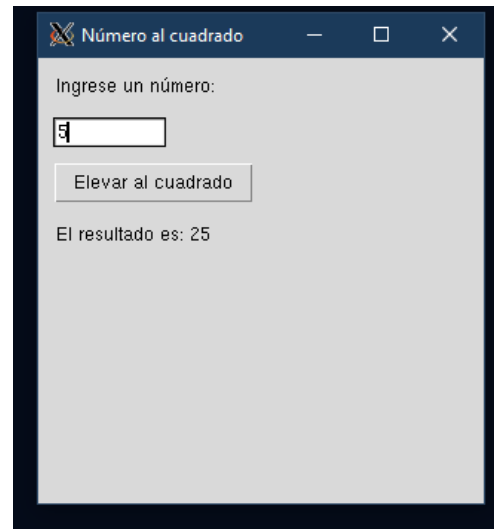
Ejercicios

Ejercicio 1: Mostrar una ventana y en su interior dos botones y una label. El label muestra inicialmente el valor 1. Cada uno de los botones permiten incrementar o decrementar en uno el contenido de la label.



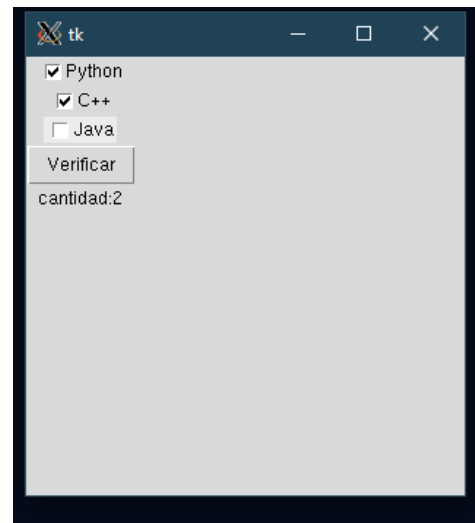


Ejercicio 2: Confeccionar una aplicación que permita ingresar un entero por teclado y al presionar un botón muestre dicho valor elevado al cuadrado en una Label.



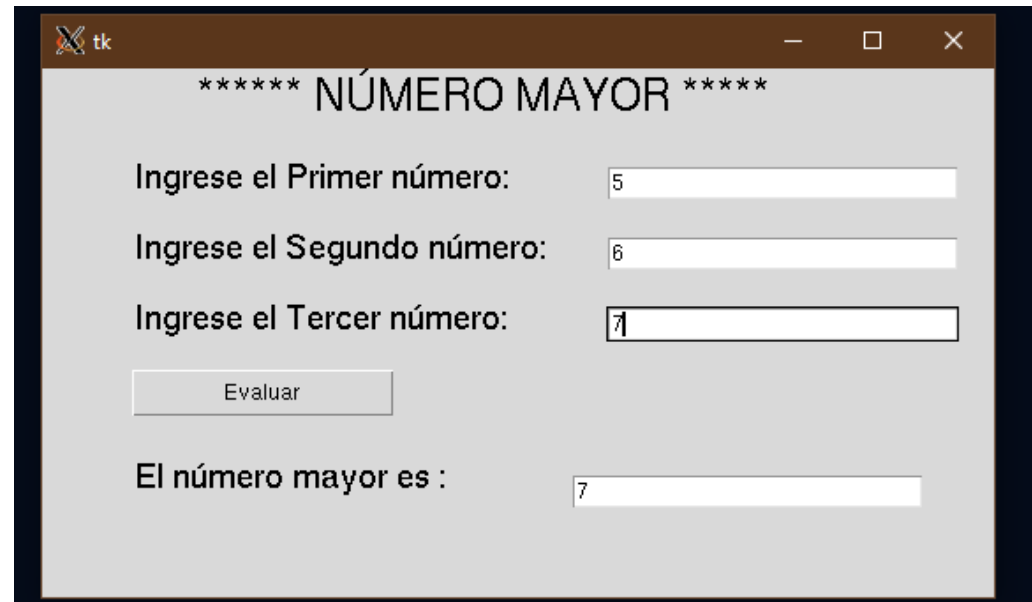


Ejercicio3: Mostrar una ventana y en su interior tres controles de tipo Checkbutton cuyas etiquetas correspondan a distintos lenguajes de programación. Cuando se presione un botón mostrar en una Label la cantidad de Checkbutton que se encuentran chequeados.





Ejercicio4: Mostrar una ventana que permita ingresar por teclado 3 números y mostrar que numero es mayor.



***** NÚMERO MAYOR *****

Ingrese el Primer número:

Ingrese el Segundo número:

Ingrese el Tercer número:

El número mayor es :



Referencias Bibliográficas

Webs:

- ❖ <https://www.adictosaltrabajo.com/2020/06/30/interfaces-graficas-en-python-con-tkinter/>
- ❖ <https://docs.python.org/3/library/tkinter.html>
- ❖ <https://docs.hektorprofe.net/python/interfaces-graficas-con-tkinter/>