¿Qué sigue?

¡Felicitaciones! Hemos llegado al final de esta pequeña introducción a Python. Hemos hecho un recorrido por los principales conceptos del lenguaje y las herramientas que ofrece. ¡Pero esto es solo el comienzo! Lo mejor para seguir adentrándonos en el mundo de Python es empezar a practicar. Si tienes algún proyecto en mente por el cual te has decidido a aprender el lenguaje, ¡ya es hora de empezarlo! No habrá mejor manera de reconocer cuáles son tus falencias y en qué cuestiones debes profundizar tu aprendizaje.

Dos grandes ramas en las que se escinde el desarrollo es en aplicaciones de escritorio y aplicaciones web. Te daré algunos consejos para que, si es tu caso, puedas tener una idea para comenzar con alguno de ellos.

Aplicaciones de escritorio

Python incluye en la librería estándar un paquete llamado [tkinter](https://recursospython.com/guias-y-manuales/introduccion-a-tkinter/), una librería gráfica pensada para aplicaciones medianas y pequeñas, y por ella razón ideal para empezar.

Otras soluciones más grandes y un tanto más complejas incluyen [PyQt](https://riverbankcomputing.com/software/pyqt/intro)/[PySide](https://wiki.qt.io/PySide) (paquetes de Python para la librería Qt), [wxPython](https://wxpython.org/) (para wxWidgets) y [PyGTK](http://pygtk.org/) (GTK+).

Si bien Python es un lenguaje interpretado, puedes empaquetar una aplicación de escritorio para que pueda ser ejecutada sin la necesidad que los usuarios tengan el intérprete instalado..

Aplicaciones web

Python resulta un lenguaje ideal para crear aplicaciones web, de hecho, es utilizado por los gigantes más importantes de internet. Lo más común es servirte de alguno de los muchos *web frameworks* que facilitan el desarrollo. El más conocido sea probablemente [Django](https://www.djangoproject.com/). Pero para comenzar te puedo recomendar [Flask](http://flask.pocoo.org/) o [web2py](http://www.web2py.com/), que son más amigables y su potencial es igualmente grande. Otra solución muy potente es [Pyramid](https://trypyramid.com/).

El desarrollo de aplicaciones web en Python difiere de otros lenguajes como PHP o Perl. En ellos se acostumbra a emplear un protocolo bastante primitivo llamado CGI. En Python, en cambio, se utiliza uno más nuevo y potente llamado WSGI. No obstante, dado que todo *hosting* compartido que utilice Linux tendrá un intérprete de Python, resultará útil también conocer cómo generar *scripts* que puedan ser ejecutados en la web.

**Qué es Tkinter: una librería de Python**

Tkinter es una librería del lenguaje de programación Python y funciona para la creación y el desarrollo de aplicaciones de escritorio. Esta librería facilita el posicionamiento y desarrollo de una [interfaz gráfica](https://keepcoding.io/blog/que-son-las-interfaces-de-usuario-componentes/) de escritorio con Python. **Tkinter es el paquete estándar de Python para interactuar con Tk.**

De acuerdo a la documentación de Python, [**TK**](https://docs.python.org/es/3/library/tkinter.html#module-tkinter)**se describe a sí mismo como el único *toolkit* o kit de herramientas para el desarrollo de una interfaz gráfica de usuario**(GUI) que funciona en todos los sistemas operativos, es decir, funciona en Windows, Mac OS y Linux. Además, está diseñado y preparado para lenguajes dinámicos con un alto nivel, como pueden ser Tcl, Ruby, Perl o Python, entre otros.

Por tanto, con su uso **no es necesario que accedas a las rutinas de bajo nivel** de cualquiera de los sistemas operativos para controlar gráficamente la pantalla de las interfaces de usuario que piensas crear. Saber qué es Tkinter te aporta mucha facilidad en el desarrollo de tus aplicaciones.

**Elementos creados con Tkinter**

***Ventana de aplicación***

Con los conceptos de la librería de Tkinter es posible realizar la venta de tu aplicación. Una **ventana de aplicación es un lienzo donde se dispondrán una serie de controles.**

***Widgets***

En TK a los controles de las ventanas de aplicación se les llama *widgets*. Estos se encargan de realizar las acciones de nuestra aplicación. Pueden ser **cuadros de texto**(donde el usuario podrá escribir), **etiquetas de texto** (el programa comunicará texto que no se podrá modificar) o **botones**, entre otros. Los controles tienen una posición y estilo que pueden modificarse. Además, responderán acciones del usuario, es decir, **los eventos gestionados.**

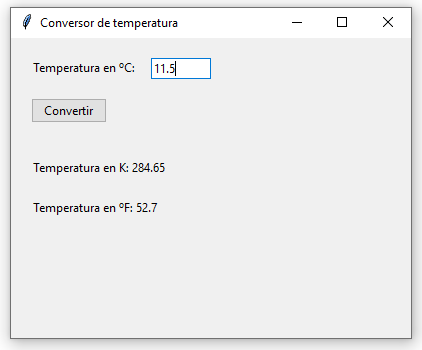
***Más ventanas***

Se utilizan en programas **donde se necesitan ventanas diferente a la ventana de aplicación.**Funcionan para varios objetivos, como mensajes de aplicación o ventanas hijas para dar de alta a un usuario, entre otros.

## Introducción a Tcl/Tk (tkinter)

Tk 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](https://www.recursospython.com/guias-y-manuales/generar-archivo-ejecutable-con-cx_freeze-pyinstaller-y-py2exe/), que se integran muy bien con Tk.

Aquí en Recursos Python tenemos una importante colección de artículos y códigos de fuentes relativos a Tk categorizados bajo la etiqueta [tkinter](https://www.recursospython.com/tag/tkinter/), que cubren una parte importante de la biblioteca. Pero puesto que todos esos artículos suponen un mínimo conocimiento del funcionamiento y de algunos términos técnicos de Tk y del desarrollo de aplicaciones de escritorio, decidimos escribir este artículo como punto de partida y de referencia de todo el resto. Con este propósito, a continuación crearemos una pequeña aplicación que permite convertir un valor de temperatura expresado en grados Celsius a valores en Kelvin y grados Fahrenheit, como lo ilustra la siguiente imagen.



Para comenzar, el primer paso para cualquier aplicación de Tk es importar los módulos correspondientes:

1. **import** tkinter **as** tk
2. **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. Para las diferencias entre tk y ttk, véase [este artículo](https://www.recursospython.com/guias-y-manuales/apariencia-y-estilos-de-los-controles-en-tkinter/).

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

1. ventana = tk.Tk()
2. ventana.mainloop()

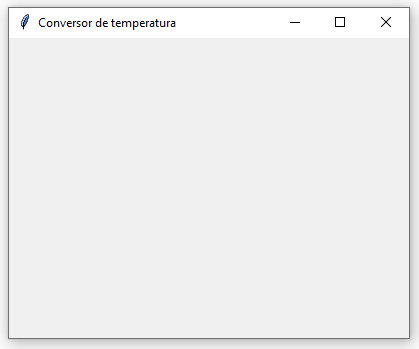
En la primera línea creamos una instancia de Tk, que se ocupa de crear la ventana principal y de iniciar internamente un intérprete de Tcl y Tk, cosa que explica el nombre de la clase. En efecto, sería más claro si el nombre fuese tk.MainWindow o similar. Con propósitos didácticos hemos llamado ventana a la instancia en cuestión, pero la convención que habitúan seguir los programadores de Tk es nombrarla root (raíz, pues es el objeto del cual nacerá el resto de la interfaz).

La segunda línea ejecuta el método mainloop(), que es el bucle principal del programa. Todas las aplicaciones de escritorio (en Tk o en cualquier otra herramienta afín) 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. Este es un dato importante a tener en cuanta siempre que atinemos a ubicar código debajo de la llamada al mainloop(). 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().

1. ventana = tk.Tk()
2. ventana.title("Conversor de temperatura")
3. ventana.config(width=400, height=300)
4. ventana.mainloop()

En cualquier método, función o clase de Tk donde se requiera especificar un tamaño, será vía los argumentos width (ancho) y height (alto), indicados en píxeles. Así, nuestra ventana tendrá al iniciarse un ancho de 400 píxeles y un alto de 300 píxeles.

[](https://www.recursospython.com/wp-content/uploads/2021/08/conversor-de-temperatura-ventana-tkinter.png)

Ahora bien, tenemos la ventana configurada y debemos empezar a llenarla de botones, cajas de texto, etiquetas, menús, casillas de verificación, etc. A cada uno de estos componentes que podemos incluir en nuestra interfaz y con los cuales el usuario puede interactuar se lo conoce como *control* o *widget*. Puedes ver una lista completa de los controles que ofrece Tk en [este artículo](https://www.recursospython.com/guias-y-manuales/apariencia-y-estilos-de-los-controles-en-tkinter/). En términos generales podemos decir que hay dos formas de organizar el código relativo a la creación de los controles: con orientación a objetos o sin orietanción a objetos. Utilizar orientación a objetos suele ser útil para las aplicaciones más grandes y con interfaces de usuario complejas. Para aplicaciones pequeñas y medianas, prescindir de la orietanción a objetos es pertinente. En lo que sigue del artículo cada bloque de código lo presentaremos en sus dos versiones.

Empecemos por mostrar un mensaje en la ventana que indique al usuario que debe ingresar la temperatura en grados Celsius, para lo cual podemos usar un control o *widget* llamado etiqueta. La etiqueta está representada en Tk por la clase ttk.Label.

* Sin orientación a objetos
* Con orientación a objetos

1. **import** tkinter **as** tk
2. **from** tkinter **import** ttk
3. ventana = tk.Tk()
4. ventana.title("Conversor de temperatura")
5. ventana.config(width=400, height=300)
6. etiqueta\_temp\_celsius = ttk.Label(text="Temperatura en ºC:")
7. etiqueta\_temp\_celsius.place(x=20, y=20)
8. 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. Para una explicación pormenorizada sobre place() y los otros métodos para posicionar controles en Tk, véase [este artículo](https://www.recursospython.com/guias-y-manuales/posicionar-elementos-en-tkinter/).

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.

* Sin orientación a objetos
* Con orientación a objetos

1. **import** tkinter **as** tk
2. **from** tkinter **import** ttk
3. ventana = tk.Tk()
4. ventana.title("Conversor de temperatura")
5. ventana.config(width=400, height=300)
6. etiqueta\_temp\_celsius = ttk.Label(text="Temperatura en ºC:")
7. etiqueta\_temp\_celsius.place(x=20, y=20)
8. caja\_temp\_celsius = ttk.Entry()
9. caja\_temp\_celsius.place(x=140, y=20, width=60)
10. boton\_convertir = ttk.Button(text="Convertir")
11. boton\_convertir.place(x=20, y=60)
12. 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, lo valores en Kelvin y grados Fahrenheit. Agreguémoslas:

* Sin orientación a objetos
* Con orientación a objetos

1. **import** tkinter **as** tk
2. **from** tkinter **import** ttk
3. ventana = tk.Tk()
4. ventana.title("Conversor de temperatura")
5. ventana.config(width=400, height=300)
6. etiqueta\_temp\_celsius = ttk.Label(text="Temperatura en ºC:")
7. etiqueta\_temp\_celsius.place(x=20, y=20)
8. caja\_temp\_celsius = ttk.Entry()
9. caja\_temp\_celsius.place(x=140, y=20, width=60)
10. boton\_convertir = ttk.Button(text="Convertir")
11. boton\_convertir.place(x=20, y=60)
12. etiqueta\_temp\_kelvin = ttk.Label(text="Temperatura en K: n/a")
13. etiqueta\_temp\_kelvin.place(x=20, y=120)
14. etiqueta\_temp\_fahrenheit = ttk.Label(text="Temperatura en ºF: n/a")
15. etiqueta\_temp\_fahrenheit.place(x=20, y=160)
16. ventana.mainloop()

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 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().

**Nuestro código final seria el siguiente**

1. **import** tkinter **as** tk
2. **from** tkinter **import** ttk
3. **def** convertir\_temp():
4. temp\_celsius = float(caja\_temp\_celsius.get())
5. temp\_kelvin = temp\_celsius + 273.15
6. temp\_fahrenheit = temp\_celsius\*1.8 + 32
7. etiqueta\_temp\_kelvin.config(text=f"Temperatura en K: {temp\_kelvin}")
8. etiqueta\_temp\_fahrenheit.config(
9. text=f"Temperatura en ºF: {temp\_fahrenheit}")
10. ventana = tk.Tk()
11. ventana.title("Conversor de temperatura")
12. ventana.config(width=400, height=300)
13. etiqueta\_temp\_celsius = ttk.Label(text="Temperatura en ºC:")
14. etiqueta\_temp\_celsius.place(x=20, y=20)
15. caja\_temp\_celsius = ttk.Entry()
16. caja\_temp\_celsius.place(x=140, y=20, width=60)
17. boton\_convertir = ttk.Button(text="Convertir", command=convertir\_temp)
18. boton\_convertir.place(x=20, y=60)
19. etiqueta\_temp\_kelvin = ttk.Label(text="Temperatura en K: n/a")
20. etiqueta\_temp\_kelvin.place(x=20, y=120)
21. etiqueta\_temp\_fahrenheit = ttk.Label(text="Temperatura en ºF: n/a")
22. etiqueta\_temp\_fahrenheit.place(x=20, y=160)
23. ventana.mainloop()

**Íconos en ventanas de Tk (tkinter)**

Tanto las ventanas principales (creadas vía la clase tk.Tk) como [las ventanas secundarias (tk.Toplevel)](https://recursospython.com/guias-y-manuales/ventanas-secundarias-tkinter/) tienen por defecto un ícono con el logo de Tcl/Tk. Configurar las ventanas de una aplicación de escritorio con íconos propios le dará a nuestro producto un aire más profesional. El ícono de una ventana también suele ser mostrado por el sistema operativo en la barra de tareas o aplicaciones. En este artículo analizaremos las diversas formas de hacerlo en las distintas plataformas soportadas por Tk (Windows, Linux y Mac).

Las ventanas de Tk proveen tres métodos para configurar un ícono:

* iconbitmap()
* iconmask()
* iconphoto()

La función iconmask() tiene el objetivo de permitir áreas transparentes en el ícono de nuestra ventana. Sin embargo, hace tiempo que los diversos formatos de imagen (PNG, GIF, JPG, etc.) y de íconos (especialmente .ico) soportan transparencia. Por lo tanto, en la mayoría de los casos iconmask() es innecesaria. Si tenemos el ícono de nuestra aplicación en un archivo de imagen, la forma más sencilla de cargarlo en la ventana es vía iconphoto():

1. **import** tkinter **as** tk
2. ventana = tk.Tk()
3. ventana.title("Ventana con ícono")
4. ventana.geometry("300x200")
5. # Cargar el archivo de imagen desde el disco.
6. icono = tk.PhotoImage(file="icon-16.png")
7. # Establecerlo como ícono de la ventana.
8. ventana.iconphoto(True, icono)
9. ventana.mainloop()

uedes descargar nuestros íconos de prueba ([clic aquí](https://recursospython.com/wp-content/uploads/2022/09/iconos.zip)) para correr el código de este artículo. El archivo comprimido contiene:

| **Archivo** | **Vista previa** | **Observaciones** |
| --- | --- | --- |
| icon-16.png | <https://recursospython.com/wp-content/uploads/2022/09/icon-16.png> | 16×16 px |
| icon-32.png | <https://recursospython.com/wp-content/uploads/2022/09/icon-32.png> | 32×32 px |
| icon.ico |  | Contiene los dos anteriores. Solo Windows. |

Los íconos deben estar en la misma carpeta que nuestro programa o, mejor dicho, en el directorio actual de trabajo.

El funcionamiento del código es sencillo: cargamos un archivo de imagen vía la clase tk.PhotoImage y luego la asignamos como ícono de la ventana a través del método iconphoto(). El primer argumento es un booleano que indica si ese mismo ícono debe aplicarse a [ventanas secundarias](https://recursospython.com/guias-y-manuales/ventanas-secundarias-tkinter/). Este procedimiento de cargar el ícono de la ventana a partir de un archivo de imagen es soportado por todos los sistemas operativos.

Además de mostrarse en la ventana, el ícono también suele mostrarse en la barra de tareas o aplicaciones. Por lo general, el tamaño adecuado para el ícono de la ventana es 16×16 píxeles. Llamaremos a esta imagen el «ícono chico». En cambio, en la barra de tareas se muestra un «ícono grande», que en Windows suele ser una imagen de 32×32. Estos valores pueden cambiar, sin embargo, acorde a la configuración de pantalla de cada sistema. Puesto que hemos cargado un ícono chico, veremos que el sistema operativo agranda automáticamente la imagen de 16×16 en la barra de tareas, produciendo un efecto pixelado:

Si cargamos un ícono grande en lugar de un ícono chico, se verá bien en la barra de tareas, pero igualmente pixelado en la ventana. Por fortuna, el método iconphoto() puede recibir dos íconos de tamaños distintos (uno chico y otro grande) para mostrar correctamente en un caso y otro:

Los tamaños de las imágenes pueden variar, siempre y cuando haya un ícono más grande que el otro. El ícono chico podría ser de 24×24 y el grande de 64×64, o cualquier otra combinación. Los tamaños ideales para cada contexto (ventana y barra de tareas o aplicaciones) dependen del sistema operativo y, en distribuciones de Linux, de cada distribución en particular. En Ubuntu, por ejemplo, el tamaño ideal para el ícono grande es 64×64. En Windows, por lo general los tamaños adecuados son 16×16 y 32×32, tal como los venimos usando.

Ahora bien, en Windows podemos recurrir al método iconbitmap() para establecer el ícono de la ventana a partir de un archivo de extensión .ico. Este formato de Microsoft es más práctico porque permite tener múltiples imágenes de diversos tamaños (16×16, 32×32, 64×64, etc.) incluidos en un solo archivo .ico. Algunos programas para crear y editar este tipo de archivos son [Greenfish Icon Editor](http://greenfishsoftware.org/gfie.php) (gratuito) e [icofx](https://icofx.ro/). Así, por ejemplo, el siguiente código establece el archivo icon.ico (disponible en la descarga más arriba) como ícono de la ventana:

1. **import** tkinter **as** tk
2. ventana = tk.Tk()
3. ventana.title("Ventana con ícono")
4. ventana.geometry("300x200")
5. ventana.iconbitmap("icon.ico")
6. ventana.mainloop()

En Windows, el método iconbitmap() también acepta como argumento, además de un archivo .ico, un archivo .exe cuyo ícono quiera emplearse como ícono de la ventana. Puesto que sys.executable contiene la ruta de nuestro ejecutable, pasándolo como argumento logramos que siempre se cargue el ícono desde el ejecutable, sin necesidad de distribuir archivos .ico o .png aparte. Nótese que este procedimiento implica que cuando el código se ejecuta desde el archivo .py, la ventana recibe el ícono de Python, puesto que sys.executable contiene en ese caso la ruta del intérprete.

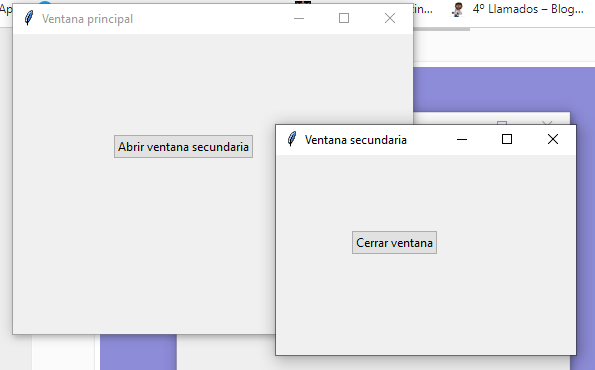
## Ventanas secundarias en Tcl/Tk (tkinter)

Las aplicaciones de escritorio pueden estar compuestas por más de una ventana. La ventana principal se crea a partir de la clase tk.Tk y controla el ciclo de vida de la aplicación. Ventanas secundarias, también conocidas como *popups*, pueden crearse al iniciar la aplicación o en respuesta a un evento (por ejemplo, la presión de un botón) vía la clase tk.Toplevel. Cuando el usuario cierra la ventana principal, todas las ventanas secundarias asimismo se cierran, finalizando la ejecución del programa. No obstante, las ventanas secundarias se pueden abrir y cerrar múltiples veces durante el ciclo de vida de la aplicación.

El siguiente código crea una aplicación con una ventana principal y una ventana secundaria. La ventana principal contiene un botón para abrir la ventana secundaria. Esta, a su vez, tiene otro botón para cerrarse.

1. **import** tkinter **as** tk
2. **from** tkinter **import** ttk
3. **def** abrir\_ventana\_secundaria():
4. # Crear una ventana secundaria.
5. ventana\_secundaria = tk.Toplevel()
6. ventana\_secundaria.title("Ventana secundaria")
7. ventana\_secundaria.config(width=300, height=200)
8. # Crear un botón dentro de la ventana secundaria
9. # para cerrar la misma.
10. boton\_cerrar = ttk.Button(
11. ventana\_secundaria,
12. text="Cerrar ventana",
13. command=ventana\_secundaria.destroy
14. )
15. boton\_cerrar.place(x=75, y=75)
16. # Crear la ventana principal.
17. ventana\_principal = tk.Tk()
18. ventana\_principal.config(width=400, height=300)
19. ventana\_principal.title("Ventana principal")
20. # Crear un botón dentro de la ventana principal
21. # que al ejecutarse invoca a la función
22. # abrir\_ventana\_secundaria().
23. boton\_abrir = ttk.Button(
24. ventana\_principal,
25. text="Abrir ventana secundaria",
26. command=abrir\_ventana\_secundaria
27. )
28. boton\_abrir.place(x=100, y=100)
29. ventana\_principal.mainloop()

Al tener dos ventanas diferentes, siempre que creamos un control (trátese de un botón o de cualquier otro), debemos especificar su ventana padre (esto es, la ventana dentro de la cual se encuentra) como primer argumento. El boton\_abrir se encuentra dentro de la ventana principal, de ahí que en la línea 28 se pase el objeto ventana\_principal como primer argumento. Lo mismo ocurre con el boton\_cerrar y la ventana secundaria en la línea 13. El resultado es el siguiente:



Ahora bien, aunque esta forma de organizar el código puede resultar útil para pequeñas aplicaciones, una mejor solución consiste en crear una clase por cada ventana. Así, el código anterior en su versión orientada a objetos se vería más o menos de este modo:

1. **import** tkinter **as** tk
2. **from** tkinter **import** ttk
3. **class** VentanaSecundaria(tk.Toplevel):
4. **def** \_\_init\_\_(self, \*args, \*\*kwargs):
5. super().\_\_init\_\_(\*args, \*\*kwargs)
6. self.config(width=300, height=200)
7. self.title("Ventana secundaria")
8. self.boton\_cerrar = ttk.Button(
9. self,
10. text="Cerrar ventana",
11. command=self.destroy
12. )
13. self.boton\_cerrar.place(x=75, y=75)
14. self.focus()
15. self.grab\_set()
16. **class** VentanaPrincipal(tk.Tk):
17. **def** \_\_init\_\_(self, \*args, \*\*kwargs):
18. super().\_\_init\_\_(\*args, \*\*kwargs)
19. self.config(width=400, height=300)
20. self.title("Ventana principal")
21. self.boton\_abrir = ttk.Button(
22. self,
23. text="Abrir ventana secundaria",
24. command=self.abrir\_ventana\_secundaria
25. )
26. self.boton\_abrir.place(x=100, y=100)
27. **def** abrir\_ventana\_secundaria(self):
28. self.ventana\_secundaria = VentanaSecundaria()
29. ventana\_principal = VentanaPrincipal()
30. ventana\_principal.mainloop()

Esta implementación tiene el beneficio de que los controles y métodos de ambas ventanas están encapsulados dentro de sus respectivos objetos (ventana\_principal y ventana\_secundaria), evitando la colisión de nombres y reduciendo el uso de objetos globales. Las clases podrían incluso estar en módulos diferentes: es un patrón común en el desarrollo de aplicaciones de escritorio colocar cada ventana en un archivo de código de fuente propio.

Otras funcionalidades son también más fáciles de implementar con esta distribución de las ventanas en clases. Por ejemplo, ¿qué ocurre si el usuario presiona dos o más veces el boton\_abrir? Si la ventana secundaria no es modal (o sea, no se ha llamado a grab\_set()), se permitirá que el usuario abra tantas ventanas secundarias como clics haya hecho en el botón. Esto es generalmente un efecto indeseable, por lo cual es útil añadir una restricción para que la VentanaSecundaria no se abra más de una vez al mismo tiempo.

Por otro lado, es habitual querer acceder a un objeto contenido en una ventana secundaria desde la ventana principal. Por ejemplo, si quisiéramos crear una ventana secundaria para que el usuario ingrese su nombre y luego necesitáramos acceder al nombre ingresado para mostrarlo en una etiqueta en la ventana principal:

import tkinter as tk

from tkinter import ttk

class VentanaNombre(tk.Toplevel):

def \_\_init\_\_(self, \*args, callback=None, \*\*kwargs):

super().\_\_init\_\_(\*args, \*\*kwargs)

# callback es una función que esta ventana llamará

# una vez presionado el botón para comunicarle el nombre

# ingresado a la ventana padre.

self.callback = callback

self.config(width=300, height=90)

# Deshabilitar el botón para maximizar la ventana.

self.resizable(0, 0)

self.title("Ingresa tu nombre")

self.caja\_nombre = ttk.Entry(self)

self.caja\_nombre.place(x=20, y=20, width=260)

self.boton\_listo = ttk.Button(

self,

text="¡Listo!",

command=self.boton\_listo\_presionado

)

self.boton\_listo.place(x=20, y=50, width=260)

self.focus()

self.grab\_set()

def boton\_listo\_presionado(self):

# Obtener el dato ingresado y llamar a la función

# especificada al crear esta ventana.

self.callback(self.caja\_nombre.get())

# Cerrar la ventana.

self.destroy()

class VentanaPrincipal(tk.Tk):

def \_\_init\_\_(self, \*args, \*\*kwargs):

super().\_\_init\_\_(\*args, \*\*kwargs)

self.config(width=400, height=300)

self.title("Ventana principal")

self.boton\_solicitar\_nombre = ttk.Button(

self,

text="Solicitar nombre",

command=self.solicitar\_nombre

)

self.boton\_solicitar\_nombre.place(x=50, y=50)

self.etiqueta\_nombre = ttk.Label(

self,

text="Aún no has ingresado ningún nombre."

)

self.etiqueta\_nombre.place(x=50, y=150)

def solicitar\_nombre(self):

# Crear la ventana secundaria y pasar como argumento

# la función en la cual queremos recibir el dato

# ingresado.

self.ventana\_nombre = VentanaNombre(

callback=self.nombre\_ingresado

)

def nombre\_ingresado(self, nombre):

# Esta función es invocada cuando el usuario presiona el

# botón "¡Listo!" de la ventana secundaria. El dato

# ingresado estará en el argumento "nombre".

self.etiqueta\_nombre.config(

text="Ingresaste el nombre: " + nombre

)

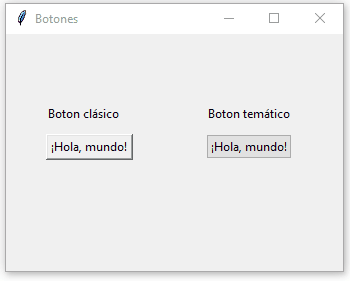
ventana\_principal = VentanaPrincipal()

ventana\_principal.mainloop()

**Apariencia y estilos de los controles en Tcl/Tk (tkinter)**

Ya tenemos nuestra aplicación de escritorio escrita en Python con Tk y ahora queremos hacer algunos cambios a la apariencia de la interfaz (botones, etiquetas, cajas de texto, etc.). Por defecto, todos los controles de una aplicación de tkinter tienen una apariencia determinada, que en la mayoría de los casos es bastante aceptable ya que conforma con los estándares del sistema operativo. No obstante, tenemos la posibilidad de alterar la apariencia por defecto de cualquier control (por ejemplo, poner en rojo el color del texto de un botón), de dos formas diferentes.

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 (que explicaremos en este artículo) 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).



(La apariencia de los botones varía, desde luego, de un sistema operativo a otro).

Cuando queremos crear un botón podemos elegir cualquiera de las dos clases; pero, además, el módulo ttk (de ***t****hemed****tk***) incluye controles nuevos que no tienen correlativos en los controles clásivos, y viceversa. He aquí la lista completa de controles:

| **Control clásico** | **Control temático** |
| --- | --- |
| [tk.Button](https://www.recursospython.com/guias-y-manuales/boton-button-en-tkinter/) | [ttk.Button](https://www.recursospython.com/guias-y-manuales/boton-button-en-tkinter/) |
| tk.Canvas |  |
|  | [ttk.Combobox](https://www.recursospython.com/guias-y-manuales/lista-desplegable-combobox-en-tkinter/) |
|  | [ttk.Checkbutton](https://www.recursospython.com/guias-y-manuales/checkbox-checkbutton-en-tcltk-tkinter/) |
| tk.Entry | [ttk.Entry](https://www.recursospython.com/guias-y-manuales/caja-de-texto-entry-tkinter/) |
| tk.Frame | ttk.Frame |
| tk.Label | ttk.Label |
|  | ttk.LabeledScale |
| tk.LabelFrame | ttk.LabelFrame |
| [tk.Listbox](https://www.recursospython.com/guias-y-manuales/lista-listbox-en-tkinter/) |  |
| [tk.Menu](https://recursospython.com/guias-y-manuales/barra-de-menu-tkinter/) |  |
| tk.Menubutton | ttk.Menubutton |
| tk.Message |  |
|  | [ttk.Notebook](https://www.recursospython.com/guias-y-manuales/panel-de-pestanas-notebook-tkinter/) |
| tk.OptionMenu | ttk.OptionMenu |
| tk.PanedWindow | ttk.PanedWindow |
|  | [ttk.Progressbar](https://www.recursospython.com/guias-y-manuales/barra-de-progreso-progressbar-tcltk-tkinter/) |
| tk.Radiobutton | ttk.Radiobutton |
| tk.Scale | ttk.Scale |
| tk.Scrollbar | ttk.Scrollbar |
|  | ttk.Separator |
|  | ttk.Sizegrip |
| tk.Spinbox | [ttk.Spinbox](https://www.recursospython.com/guias-y-manuales/caja-de-texto-numerica-spinbox-en-tkinter/) |
| tk.Text |  |
|  | [ttk.Treeview](https://www.recursospython.com/guias-y-manuales/vista-de-arbol-treeview-en-tkinter/) |

Cuando un control está disponible tanto en su versión clásica como temática, siempre conviene la temática. Las razones ya las mencionamos: la estética y el sistema de estilos son mejores y más modernos.

Veamos, ahora sí, cómo cambiar la apariencia de ambos tipos de controles. Empecemos por los controles temáticos.

**Controles temáticos**

El concepto central para comprender cómo cambiar la estética de los controles temáticos es el de **estilo**. Un estilo es una clase que contiene la información de la apariencia de un *tipo*de control. Para que un control en particular reciba la apariencia de un estilo, debemos indicar el nombre del estilo al momento de crear el control. Por defecto todos los controles temáticos tienen asignado un estilo.

Por ejemplo, creemos una ventana con un botón temático:

1. **from** tkinter **import** ttk
2. **import** tkinter **as** tk
3. ventana = tk.Tk()
4. ventana.config(width=300, height=200)
5. ventana.title("Estilos en Tk")
6. boton = ttk.Button(text="¡Hola, mundo!")
7. boton.place(x=40, y=50)
8. ventana.mainloop()

Fondo gris, color de texto negro, contorno gris oscuro, espaciado de dos o tres píxeles, etc., son todas propiedades del estilo que tiene aplicado por defecto este botón. El nombre del estilo (todos los estilos deben tener un nombre) aplicado por defecto a los botones es TButton. Por lo general, los nombres de los estilos por defecto están constituidos por una «T» seguida del nombre del control, aunque hay ciertas excepciones. He aquí la lista completa:

| **Control temático** | **Estilo por defecto** |
| --- | --- |
| ttk.Button | TButton |
| ttk.Checkbutton | TCheckbutton |
| ttk.Combobox | TCombobox |
| ttk.Entry | TEntry |
| ttk.Frame | TFrame |
| ttk.Label | TLabel |
| ttk.LabeledScale | TFrame |
| ttk.LabelFrame | TLabelframe |
| ttk.Menubutton | TMenubutton |
| ttk.Notebook | TNotebook |
| ttk.PanedWindow | TPanedwindow |
| ttk.Progressbar | TProgressbar |
| ttk.Spinbox | TSpinbox |
| ttk.Treeview | Treeview (sin prefjio) |

Desde nuestro código de Python podemos cambiar las propiedades de cada estilo. Así, por ejemplo, si modificamos el estilo TButton para que el color del texto sea rojo, todos los botones creados a partir de la clase ttk.Button tendrán el texto rojo.

1. s = ttk.Style()
2. s.configure("TButton", foreground="#ff0000")
3. boton = ttk.Button(text="¡Hola, mundo!")
4. boton.place(x=40, y=50)

Estos son los conceptos principales del sistema de estilos de Tk. Existen otros que se podrán estudiar en el caso particular de cada control. Veamos ahora cómo modificar la apariencia de los controles clásicos.

#### Controles clásicos

Modificar la apariencia de un control clásico es considerablemente más sencillo que usar el sistema de estilos que acabamos de ver, pero se trata de un método más desordenado y que no incentiva la reutilización de código. Sencillamente se pasan como argumento las propiedades que regulan la apariencia de un control al momento de crearlo.

1. **import** tkinter **as** tk
2. ventana = tk.Tk()
3. ventana.config(width=300, height=200)
4. ventana.title("Apariencia de controles clásicos")
5. boton = tk.Button(text="¡Hola, mundo!", foreground="#ff0000")
6. boton.place(x=40, y=50)
7. ventana.mainloop()

haga este ejemplo en otro archivo

1. **import** tkinter **as** tk
2. **class** BotonPeligro(tk.Button):
3. **def** \_\_init\_\_(self, \*args, \*\*kwargs):
4. # Procurar no reemplazar los argumentos provistos.
5. **if** "foreground" **not** **in** kwargs:
6. kwargs["foreground"] = "#ff0000"
7. **if** "activeforeground" **not** **in** kwargs:
8. kwargs["activeforeground"] = "#FFA500"
9. super().\_\_init\_\_(\*args, \*\*kwargs)
10. ventana = tk.Tk()
11. ventana.config(width=300, height=200)
12. ventana.title("Apariencia de controles clásicos")
13. boton = BotonPeligro(text="¡Hola, mundo!")
14. boton.place(x=40, y=50)
15. ventana.mainloop()

**Caja de texto (Entry) en Tcl/Tk (tkinter)**

Una caja de texto permite al usuario ingresar cualquier texto de una línea. En Tcl/Tk está representada a través de la clase ttk.Entry, que a su vez hereda la funcionalidad de un control más primitivo llamado tk.Entry

Para crear una caja de texto, entonces, vamos a crear una instancia de la primera clase.

1. **import** tkinter **as** tk
2. **from** tkinter **import** ttk
3. root = tk.Tk()
4. root.config(width=300, height=200)
5. # Crear caja de texto.
6. entry = ttk.Entry()
7. # Posicionarla en la ventana.
8. entry.place(x=50, y=50)
9. root.mainloop()

Para especificar la justificación del texto, esto es, si debe alinearse a la izquierda, derecha o el centro, indicamos el parámetro justify, cuyos valores pueden ser, respectivamente, tk.LEFT (por defecto), tk.RIGHT y tk.CENTER.

1. # Justificar el texto a la derecha.
2. entry = ttk.Entry(justify=tk.RIGHT)

Podemos hacer que la caja de texto muestre un glifo en particular cuando queremos ocultar el texto a simple vista; por ejemplo, si el control es empleado para escribir una contraseña.

1. # Mostrar asteriscos en lugar del texto original.
2. entry = ttk.Entry(show="\*")

A través del parámetro width indicamos el ancho del control, pero no en píxeles, sino en caracteres. Por ejemplo, el siguiente código establece un ancho suficiente para visualizar una línea de 10 caracteres completamente.

1. entry = ttk.Entry(width=10)

Si el texto excede la cantidad de caracteres especificados, el control siempre muestra una porción de él de dicho tamaño, según la posición del cursor. Puesto que el ancho de cada carácter puede variar aun en la misma fuente, el valor de width opera siempre como una aproximación y no como un valor absoluto.

La caja de texto puede iniciarse como deshabilitada (el usuario no podrá escribir en ella y aparecerá sombreada) usando el parámetro state.

1. entry = ttk.Entry(state=tk.DISABLED)

Luego podremos habilitarla nuevamente restableciendo este parámetro vía el método config().

1. entry.config(state=tk.NORMAL)

Un estado intermedio es "readonly" (solo lectura), en el que el usuario visualiza el control normalmente pero no puede escribir en él.

1. entry = ttk.Entry(state="readonly")

Esto es especialmente útil cuando se quiere prohibir la inserción de texto pero mantener la opción de copiado o los menús contextuales.

Una caja de texto puede recibir el foco cuando si el usuario presiona sucesivamente la tecla Tab, desplazándose por los diversos controles de nuestra interfaz. Si queremos evitar que nuestra caja de texto pueda recibir el foco de esta manera, empleamos el argumento takefocus.

**Operaciones principales**

Ahora bien, una vez creado, las operaciones principales son las siguientes: inserción, obtención, selección y eliminación de texto. Y de éstas, la más frecuente es la segunda, por cuanto nos permite saber qué escribió el usuario de nuestra aplicación. Para las operaciones de copiar, cortar y pegar, véase este artículo.

Empecemos, entonces, por cómo obtener lo que el usuario ha escrito en la caja de texto llamando al método get().

1. **print**(entry.get())

Por ejemplo, el siguiente código imprime en la consola el texto ingresado al presionar un botón.

1. entry.insert(0, "Hola mundo!")
2. entry.place(x=50, y=50)
3. button = ttk.Button(text="Obtener texto", command=**lambda**: **print**(entry.get()))
4. button.place(x=50, y=100)

Para añadir un texto empleamos insert(), que toma como primer argumento una posición y como segundo una cadena.

1. entry.insert(0, "¡Hola, mundo!")

Este código es similar a:

1. entry.insert(0, "¡Hola,")
2. entry.insert(6, " mundo!")

Todas las operaciones en Tk que requieran una posición aceptan la constante tk.END, que representa el final del texto. En nuestro caso, tk.END es equivalente a len(entry.get()). Por esto, el siguiente código también consigue el mismo resultado que los dos anteriores.

1. entry.insert(0, "¡Hola,")
2. entry.insert(tk.END, " mundo!")

La eliminación de un texto o una parte de él se consigue a través del método delete(), pasándole como argumentos los índices de inicio y fin.

1. # Elimina el texto completo.
2. entry.delete(0, tk.END)

De forma análoga opera select\_range(), que selecciona el texto desde una posición hasta otra. Por ejemplo, el siguiente código selecciona la palabra «mundo».

1. entry.insert(0, "¡Hola, mundo!")
2. # Seleccionar desde el carácter 7 hasta el 12.
3. entry.select\_range(7, 12)
4. # Enviar el foco a la caja de texto para hacer efectiva
5. # la selección.
6. entry.focus()

Para obtener la posición del cursor en la caja de texto, llamamos al método index() con la constante tk.INSERT.

1. **print**(entry.index(tk.INSERT))

Y para establecer la posición:

1. entry.insert(0, "¡Hola, mundo!")
2. # Ubica el cursor antes de la "m".
3. entry.icursor(7)
4. # Envía el foco.
5. entry.focus()

Tk no provee una función específica para retornar el texto seleccionado, pero haciendo uso del método index() junto con las constantes tk.SEL\_FIRST y tk.SEL\_LAST ─que retornan los índices de inicio y fin de la selección─ podemos construirla manualmente.

1. entry.insert(0, "¡Hola, mundo!")
2. **def** get\_selection():
3. # Comprobar primero que haya una selección.
4. **if** entry.select\_present():
5. # Obtener los índices del inicio y fin de la selección
6. first = entry.index(tk.SEL\_FIRST)
7. last = entry.index(tk.SEL\_LAST)
8. **print**(entry.get()[first:last])
9. **else**:
10. **print**("No hay selección.")
11. button = ttk.Button(text="Obtener selección", command=get\_selection)

(El método select\_present() retorna True o False según haya un texto seleccionado o no).

**Asociando una variable**

Cuando una caja de texto es simplemente empleada para recibir un texto ingresado por el usuario, será suficiente con llamar al método get() para obtenerlo cuando sea necesario tal como lo expusimos anteriormente. Pero si nuestro programa requiere cierta funcionalidad bilateral ─se estará tanto insertando texto como obteniendo el ingresado por el usuario─ a menudo es más conveniente asociar una variable.

Tk nos provee la clase tk.StringVar() para crear objetos que actúan como una cadena, con la excepción de que para asignarle un valor usamos el método set() y, para obtenerlo, get().

1. var = tk.StringVar()
2. var.set("¡Hola, mundo!")
3. **print**(var.get())

Ahora bien, podemos asociar una variable de estas características a una caja de texto al momento de su creación, vía el parámetro textvariable.

1. entry\_var = tk.StringVar()
2. entry = ttk.Entry(textvariable=entry\_var)

Así, llamadas a entry\_var.set() modificarán el contenido de la caja de texto y, del mismo modo, los cambios efectuados por el usuario en el control afectarán el resultado de entry\_var.get().

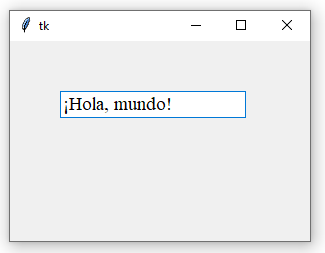
**Validaciones**

Tk permite validar el contenido de una caja de texto (por ejemplo, para solo permitir números o fechas) en el momento en que el usuario escribe. Tenemos un artículo aparte para ello: [Validar el contenido de una caja de texto en Tcl/Tk (tkinter)](https://recursospython.com/guias-y-manuales/validar-el-contenido-de-una-caja-de-texto-en-tkinter/).

**Tipo de letra**

Es posible configurar el tipo de letra de una caja de texto vía el parámetro font. Por ejemplo, el siguiente código crea una caja de texto con una tipografía de la familia *Times* y un tamaño de 14 puntos.

1. **import** tkinter **as** tk
2. **from** tkinter **import** font, ttk
3. root = tk.Tk()
4. root.config(width=300, height=200)
5. entry = ttk.Entry(font=font.Font(family="Times", size=14))
6. entry.place(x=50, y=50)
7. root.mainloop()

[](https://recursospython.com/wp-content/uploads/2018/08/ttk-entry-font.png)

El argumento font utilizado al crear la caja de texto debe recibir una instancia de la clase tkinter.font.Font. Tk soporta una gran cantidad de familias de fuentes. Para ver una lista completa de los valores posibles del argumento family, ejecútese el siguiente código en la [consola interactiva](https://tutorial.recursospython.com/consola-interactiva/):

1. >>> **import** tkinter **as** tk
2. >>> root = tk.Tk()
3. >>> font.families() # Retorna una lista de las fuentes disponibles.

Otros argumentos aceptables para la configuración de la fuente son weight, slant, overstrike y underline, que indican, respectivamente, el grosor, la inclinación, el tachado y el subrayado:

1. entry = ttk.Entry(
2. font=font.Font(
3. family="Courier",
4. size=14,
5. weight=font.BOLD, # Negrita.
6. slant=font.ITALIC, # Cursiva.
7. overstrike=True, # Tachado.
8. underline=True # Subrayado.
9. )
10. )

**Estilos**

El resto de la apariencia de una caja de texto puede configurarse a través de estilos. Para una introducción al funcionamiento de los estilos, véase [Apariencia y estilos de los controles en Tcl/Tk (tkinter)](https://recursospython.com/guias-y-manuales/apariencia-y-estilos-de-los-controles-en-tkinter/).

El siguiente código crea un estilo para una caja de texto y establece las propiedades foreground y fieldbackground, que representan el color del texto y el color de fondo, respectivamente.

1. **import** tkinter **as** tk
2. **from** tkinter **import** ttk
3. root = tk.Tk()
4. root.config(width=300, height=200)
5. style = ttk.Style()
6. style.configure(
7. "MyEntry.TEntry",
8. # Fondo rojo.
9. fieldbackground="#ff0000",
10. # Color de texto azul.
11. foreground="#0000ff"
12. )
13. entry = ttk.Entry(style="MyEntry.TEntry")
14. entry.place(x=50, y=50)
15. root.mainloop()

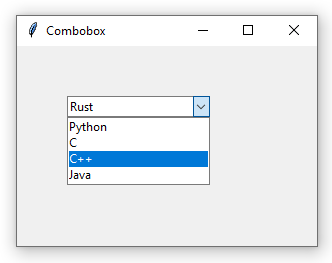
#### Estilos en el control clásico

Las aplicaciones nuevas deberían usar siempre la clase ttk.Entry. Pero si estás actualizando un antiguo código de Tk que aún utiliza el control clásico tk.Entry, al no soportar estilos deberás configurar el aspecto a través de argumentos en el momento en que se crea la caja de texto.

Para los colores del texto y del fondo, utilícense los argumentos foreground y background, respectivamente:

**Lista desplegable (Combobox) en Tcl/Tk (tkinter)**

En la jerga del desarrollo de interfaces de usuario se lo conoce como *Combobox*, pues se trata de una **comb**inación entre una [lista](https://www.recursospython.com/guias-y-manuales/lista-listbox-en-tkinter/) y una caja de texto. Así, puede actuar como una lista desplegable con determinadas opciones y eventualmente permitir al usuario escribir un valor que no se encuentre en la lista.

[](https://recursospython.com/wp-content/uploads/2017/06/combobox-tkinter.png)

El control fue introducido en Tk 8.5 y es provisto a través de la clase ttk.Combobox, la cual hereda de [ttk.Entry](https://recursospython.com/guias-y-manuales/caja-de-texto-entry-tkinter/), por lo que todas las operaciones aplicables a una [caja de texto](https://recursospython.com/guias-y-manuales/caja-de-texto-entry-tkinter/) también aplican para un *combobox*.

El siguiente código crea una simple ventana con una lista desplegable vacía.

1. **from** tkinter **import** ttk
2. **import** tkinter **as** tk
3. main\_window = tk.Tk()
4. main\_window.config(width=300, height=200)
5. main\_window.title("Combobox")
6. combo = ttk.Combobox()
7. combo.place(x=50, y=50)
8. main\_window.mainloop()

o el siguiente

1. **from** tkinter **import** ttk
2. **import** tkinter **as** tk
3. **class** Application(ttk.Frame):
4. **def** \_\_init\_\_(self, main\_window):
5. super().\_\_init\_\_(main\_window)
6. main\_window.title("Combobox")
7. self.combo = ttk.Combobox(self)
8. self.combo.place(x=50, y=50)
9. main\_window.config(width=300, height=200)
10. self.place(width=300, height=200)
11. main\_window = tk.Tk()
12. app = Application(main\_window)
13. app.mainloop()

Un *combobox* en Tk trabaja únicamente con cadenas. Cualquier otro tipo de objeto será convertido antes de ser añadido a la lista.

El método Combobox.get() retorna la opción seleccionada: ya haya sido esta seleccionada a partir de una de las opciones, ya haya sido escrita manualmente por el usuario (si no se deshabilitó esta opción vía state="readonly"). Por ejemplo, el siguiente código muestra en un [cuadro de diálogo](https://recursospython.com/guias-y-manuales/cuadros-de-dialogo-messagebox-en-tkinter/) la opción seleccionada en la lista desplegable tras presionar un botón.

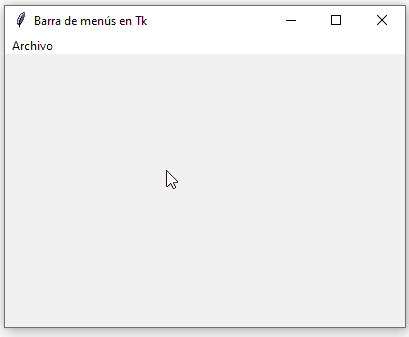
1. **from** tkinter **import** messagebox, ttk
2. **import** tkinter **as** tk
3. **def** show\_selection():
4. # Obtener la opción seleccionada.
5. selection = combo.get()
6. messagebox.showinfo(
7. message=f"La opción seleccionada es: {selection}",
8. title="Selección"
9. )
10. main\_window = tk.Tk()
11. main\_window.config(width=300, height=200)
12. main\_window.title("Combobox")
13. combo = ttk.Combobox(
14. state="readonly",
15. values=["Python", "C", "C++", "Java"]
16. )
17. combo.place(x=50, y=50)
18. button = ttk.Button(text="Mostrar selección", command=show\_selection)
19. button.place(x=50, y=100)
20. main\_window.mainloop()

En formato orientado a objeto

1. **from** tkinter **import** messagebox, ttk
2. **import** tkinter **as** tk
3. **class** Application(ttk.Frame):
4. **def** \_\_init\_\_(self, main\_window):
5. super().\_\_init\_\_(main\_window)
6. main\_window.title("Combobox")
7. self.combo = ttk.Combobox(
8. self,
9. state="readonly",
10. values=["Python", "C", "C++", "Java"]
11. )
12. self.combo.place(x=50, y=50)
13. self.button = ttk.Button(
14. text="Mostrar selección",
15. command=self.show\_selection
16. )
17. self.button.place(x=50, y=100)
18. main\_window.config(width=300, height=200)
19. self.place(width=300, height=200)
20. **def** show\_selection(self):
21. # Obtener la opción seleccionada.
22. selection = self.combo.get()
23. messagebox.showinfo(
24. message=f"La opción seleccionada es: {selection}",
25. title="Selección"
26. )
27. main\_window = tk.Tk()
28. app = Application(main\_window)
29. app.mainloop()

## Barra de menú en Tcl/Tk (tkinter)

El control tk.Menu permite añadir una barra de menús a la ventana principal (tk.Tk) o a [una ventana secundaria (tk.Toplevel)](https://recursospython.com/guias-y-manuales/ventanas-secundarias-tkinter/) en una aplicación de escritorio de Tk. Los menús de una ventana contienen un texto y/o una imagen y pueden ser asociados a funciones para responder ante la presión del usuario.



En la imagen vemos una barra de menús en la ventana principal, un menú con el título «Archivo» y un botón dentro de este menú con el texto «Nuevo», el atajo «Ctrl+N» y una imagen a modo de ícono. Para implementar una funcionalidad como esta, lo primordial es crear una barra de menús vía la clase tk.Menu y configurarla en la ventana principal.

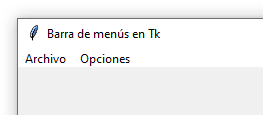
1. **import** tkinter **as** tk
2. **def** archivo\_nuevo\_presionado():
3. **print**("¡Has presionado para crear un nuevo archivo!")
4. ventana = tk.Tk()
5. ventana.title("Barra de menús en Tk")
6. ventana.config(width=400, height=300)
7. barra\_menus = tk.Menu()
8. menu\_archivo = tk.Menu(barra\_menus, tearoff=False)
9. menu\_archivo.add\_command(
10. label="Nuevo",
11. accelerator="Ctrl+N",
12. command=archivo\_nuevo\_presionado
13. )
14. barra\_menus.add\_cascade(menu=menu\_archivo, label="Archivo")
15. ventana.config(menu=barra\_menus)
16. ventana.mainloop()

Puedes descargar la imagen nuevo\_archivo.png desde [este enlace](https://recursospython.com/wp-content/uploads/2022/03/nuevo_archivo.png). El archivo debe estar en la misma carpeta que la aplicación.

1. **import** tkinter **as** tk
2. **def** archivo\_nuevo\_presionado(event=None):
3. **print**("¡Has presionado para crear un nuevo archivo!")
4. ventana = tk.Tk()
5. ventana.title("Barra de menús en Tk")
6. ventana.config(width=400, height=300)
7. barra\_menus = tk.Menu()
8. menu\_archivo = tk.Menu(barra\_menus, tearoff=False)
9. img\_menu\_nuevo = tk.PhotoImage(file="nuevo\_archivo.png")
10. menu\_archivo.add\_command(
11. label="Nuevo",
12. accelerator="Ctrl+N",
13. command=archivo\_nuevo\_presionado,
14. image=img\_menu\_nuevo,
15. # Indicar que la imagen debe aparecer a la izquierda del texto.
16. compound=tk.LEFT
17. )
18. ventana.bind\_all("<Control-n>", archivo\_nuevo\_presionado)
19. barra\_menus.add\_cascade(menu=menu\_archivo, label="Archivo")
20. ventana.config(menu=barra\_menus)
21. ventana.mainloop()

Del mismo modo, con sucesivas llamadas a add\_cascade() podemos agregar otros menús a la barra de menús. El siguiente código agrega un segundo menú con el título «Opciones».

1. **import** tkinter **as** tk
2. **def** archivo\_nuevo\_presionado(event=None):
3. **print**("¡Has presionado para crear un nuevo archivo!")
4. ventana = tk.Tk()
5. ventana.title("Barra de menús en Tk")
6. ventana.config(width=400, height=300)
7. barra\_menus = tk.Menu()
8. menu\_archivo = tk.Menu(barra\_menus, tearoff=False)
9. img\_menu\_nuevo = tk.PhotoImage(file="nuevo\_archivo.png")
10. menu\_archivo.add\_command(
11. label="Nuevo",
12. accelerator="Ctrl+N",
13. command=archivo\_nuevo\_presionado,
14. image=img\_menu\_nuevo,
15. compound=tk.LEFT
16. )
17. ventana.bind\_all("<Control-n>", archivo\_nuevo\_presionado)
18. menu\_archivo.add\_separator()
19. menu\_archivo.add\_command(label="Salir", command=ventana.destroy)
20. menu\_opciones = tk.Menu(barra\_menus, tearoff=False)
21. barra\_menus.add\_cascade(menu=menu\_archivo, label="Archivo")
22. barra\_menus.add\_cascade(menu=menu\_opciones, label="Opciones")
23. ventana.config(menu=barra\_menus)
24. ventana.mainloop()

[](https://recursospython.com/wp-content/uploads/2022/03/ventana-con-menu-opciones-tkinter.png)

Un menú puede contener botones tradicionales, como los que acabamos de usar para las opciones «Nuevo» y «Salir», o botones con casillas de verificación. El funcionamiento de estos últimos es similar al de la clase [ttk.Checkbutton](https://recursospython.com/guias-y-manuales/checkbox-checkbutton-en-tcltk-tkinter/). Los botones de menús con casilla de verificación se comportan igual que los botones normales, pero contienen además un [valor booleano](https://recursospython.com/guias-y-manuales/booleanos-operaciones-logicas-y-binarias/) que es alterado cada vez que el usuario presiona sobre ellos. Este valor booleano está representado por la presencia de una marca a la izquierda del texto del botón.

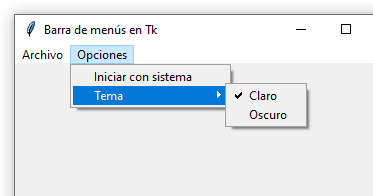
El código es el siguiente.

1. **import** tkinter **as** tk
2. **def** archivo\_nuevo\_presionado(event=None):
3. **print**("¡Has presionado para crear un nuevo archivo!")
4. **def** menu\_iniciar\_con\_sistema\_presionado():
5. **if** iniciar\_con\_sistema.get():
6. **print**("Opción establecida (iniciar con el sistema).")
7. **else**:
8. **print**("Opción deshabilitada (no iniciar con el sistema).")
9. ventana = tk.Tk()
10. ventana.title("Barra de menús en Tk")
11. ventana.config(width=400, height=300)
12. barra\_menus = tk.Menu()
13. menu\_archivo = tk.Menu(barra\_menus, tearoff=False)
14. img\_menu\_nuevo = tk.PhotoImage(file="nuevo\_archivo.png")
15. menu\_archivo.add\_command(
16. label="Nuevo",
17. accelerator="Ctrl+N",
18. command=archivo\_nuevo\_presionado,
19. image=img\_menu\_nuevo,
20. compound=tk.LEFT
21. )
22. ventana.bind\_all("<Control-n>", archivo\_nuevo\_presionado)
23. menu\_archivo.add\_separator()
24. menu\_archivo.add\_command(label="Salir", command=ventana.destroy)
25. menu\_opciones = tk.Menu(barra\_menus, tearoff=False)
26. iniciar\_con\_sistema = tk.BooleanVar()
27. menu\_opciones.add\_checkbutton(
28. label="Iniciar con sistema",
29. command=menu\_iniciar\_con\_sistema\_presionado,
30. variable=iniciar\_con\_sistema
31. )
32. barra\_menus.add\_cascade(menu=menu\_archivo, label="Archivo")
33. barra\_menus.add\_cascade(menu=menu\_opciones, label="Opciones")
34. ventana.config(menu=barra\_menus)
35. ventana.mainloop()

Como se observa en la línea 30, para crear un botón con casilla de verificación dentro de un menú se utiliza add\_checkbutton() en lugar de add\_command(). A este método es necesario pasarle una variable booleana de Tk creada vía la clase tk.BooleanVar() (líneas 29 y 33). Cada vez que el usuario presiona el botón, Tk cambia el valor booleano de la instancia iniciar\_con\_sistema y además invoca la función menu\_iniciar\_con\_sistema\_presionado(). El valor booleano de iniciar\_con\_sistema puede ser leído o establecido vía código a través de los método get() (tal como ocurre en la línea 7) y set().

Esta misma lógica tiene el método add\_radiobutton(), empleado para añadir varios botones con casilla de verificación dentro de un menú, pero que tienen una relación entre sí de tal modo que cuando uno de ellos está activado, el resto se desactiva. Por ejemplo, si queremos permitirle al usuario elegir a través de los menús de nuestra aplicación el color del tema de la interfaz entre las opciones «Claro» y «Oscuro», sería conveniente emplear un par de botones de este estilo.

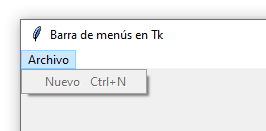
1. **import** tkinter **as** tk
2. **def** archivo\_nuevo\_presionado(event=None):
3. **print**("¡Has presionado para crear un nuevo archivo!")
4. **def** menu\_iniciar\_con\_sistema\_presionado():
5. **if** iniciar\_con\_sistema.get():
6. **print**("Opción establecida (iniciar con el sistema).")
7. **else**:
8. **print**("Opción deshabilitada (no iniciar con el sistema).")
9. **def** menu\_tema\_presionado():
10. valor\_tema = tema\_elegido.get()
11. **if** valor\_tema == 1:
12. **print**("Tema claro establecido.")
13. **elif** valor\_tema == 2:
14. **print**("Tema oscuro establecido.")
15. ventana = tk.Tk()
16. ventana.title("Barra de menús en Tk")
17. ventana.config(width=400, height=300)
18. barra\_menus = tk.Menu()
19. menu\_archivo = tk.Menu(barra\_menus, tearoff=False)
20. img\_menu\_nuevo = tk.PhotoImage(file="nuevo\_archivo.png")
21. menu\_archivo.add\_command(
22. label="Nuevo",
23. accelerator="Ctrl+N",
24. command=archivo\_nuevo\_presionado,
25. image=img\_menu\_nuevo,
26. compound=tk.LEFT
27. )
28. ventana.bind\_all("<Control-n>", archivo\_nuevo\_presionado)
29. menu\_archivo.add\_separator()
30. menu\_archivo.add\_command(label="Salir", command=ventana.destroy)
31. menu\_opciones = tk.Menu(barra\_menus, tearoff=False)
32. iniciar\_con\_sistema = tk.BooleanVar()
33. menu\_opciones.add\_checkbutton(
34. label="Iniciar con sistema",
35. command=menu\_iniciar\_con\_sistema\_presionado,
36. variable=iniciar\_con\_sistema
37. )
38. menu\_tema = tk.Menu(barra\_menus, tearoff=False)
39. tema\_elegido = tk.IntVar()
40. tema\_elegido.set(1) # Opción seleccionada por defecto ("Claro").
41. menu\_tema.add\_radiobutton(
42. label="Claro",
43. variable=tema\_elegido,
44. value=1,
45. command=menu\_tema\_presionado
46. )
47. menu\_tema.add\_radiobutton(
48. label="Oscuro",
49. value=2,
50. variable=tema\_elegido,
51. command=menu\_tema\_presionado
52. )
53. menu\_opciones.add\_cascade(menu=menu\_tema, label="Tema")
54. barra\_menus.add\_cascade(menu=menu\_archivo, label="Archivo")
55. barra\_menus.add\_cascade(menu=menu\_opciones, label="Opciones")
56. ventana.config(menu=barra\_menus)
57. ventana.mainloop()

[](https://recursospython.com/wp-content/uploads/2022/03/ventana-con-menu-radiobutton-tkinter.png)

Hay varias cosas para notar en este código. En primer lugar, creamos un nuevo menú llamado menu\_tema que fue a su vez añadido al menú de opciones vía add\_cascade() (líneas 42 y 57). Como el mismo nombre lo indica (cascade, «cascada»), puede haber menús dentro de otros menús que se despliegan en forma de cascada tal como se observa en la imagen. Segundo, dentro del menu\_tema insertamos dos botones vía los métodos add\_radiobutton(), pero a diferencia de la opción anterior insertada con add\_checkbutton(), aquí ambas opciones hacen referencia a la misma variable entera tema\_elegido. El hecho de que refieran a la misma variable le indica a Tk que esas dos opciones (aunque podrían ser más de dos) son incompatibles: cuando el usuario presiona el botón «Claro», se remueve la selección del tema «Oscuro» y viceversa. Por último, nótese que cada llamada a add\_radiobutton() incluye un argumento value que indica el valor numérico (pues la variable creada es tk.IntVar) que representa la opción que se está agregando al menú. Este valor será retornado por menu\_tema.get() cuando la opción esté seleccionada, como vemos en la función menu\_tema\_presionado().

#### Estados

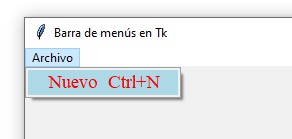
El botón de un menú puede estar habilitado o deshabilitado, igual que un botón. Cuando está desactivado, el texto y la imagen se muestran con un color diferente y el usuario no puede presionar sobre él. Los métodos add\_command(), add\_checkbutton() y add\_radiobutton() soportan el argumento state, que denota el estado del botón.



1. **import** tkinter **as** tk
2. ventana = tk.Tk()
3. ventana.title("Barra de menús en Tk")
4. ventana.config(width=400, height=300)
5. barra\_menus = tk.Menu()
6. menu\_archivo = tk.Menu(barra\_menus, tearoff=False)
7. menu\_archivo.add\_command(
8. label="Nuevo",
9. accelerator="Ctrl+N",
10. state=tk.DISABLED
11. )
12. barra\_menus.add\_cascade(menu=menu\_archivo, label="Archivo")
13. ventana.config(menu=barra\_menus)
14. ventana.mainloop()

**Estilos**

La apariencia de los botones de los menús se personaliza pasando argumentos a las funciones add\_command(), add\_checkbutton() y add\_radiobutton(). Las tres soportan cinco propiedades que establecen el tipo de letra y los colores de los botones.



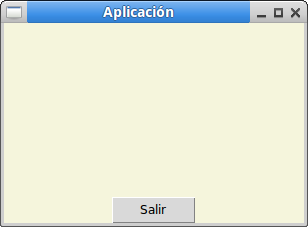
1. **import** tkinter **as** tk
2. **from** tkinter **import** font
3. ventana = tk.Tk()
4. ventana.title("Barra de menús en Tk")
5. ventana.config(width=400, height=300)
6. barra\_menus = tk.Menu()
7. menu\_archivo = tk.Menu(barra\_menus, tearoff=False)
8. menu\_archivo.add\_command(
9. label="Nuevo",
10. accelerator="Ctrl+N",
11. # Tipo de fuente.
12. font=font.Font(family="Times", size=14),
13. # Color de fondo.
14. background="#ADD8E6",
15. # Color del texto.
16. foreground="#FF0000",
17. # Color de fondo cuanto el botón tiene el foco.
18. activebackground="#32CDFF",
19. # Color del texto cuando el botón tiene el foco.
20. activeforeground="#FFFF00"
21. )
22. barra\_menus.add\_cascade(menu=menu\_archivo, label="Archivo")
23. ventana.config(menu=barra\_menus)
24. ventana.mainloop()

**BASANDOME EN OTRO TUTORIAL PARA SACARLE EL PROVECHO DE PYTHON AL MAXIMO**

Con Python [hay muchas posibilidades para programar una interfaz gráfica de usuario](https://wiki.python.org/moin/GuiProgramming) ([**GUI**](https://es.wikipedia.org/wiki/Interfaz_gr%C3%A1fica_de_usuario)) pero **Tkinter** es fácil de usar, es multiplataforma y, además, viene incluido con Python en su versión para Windows, para Mac y para la mayoría de las distribuciones GNU/Linux. Se le considera el estándar de facto en la programación GUI con Python.  
  
**Tkinter** es un [**binding**](https://es.wikipedia.org/wiki/Binding#Inform.C3.A1tica) de la biblioteca [Tcl](https://es.wikipedia.org/wiki/Tcl)/[Tk](https://es.wikipedia.org/wiki/Tk) que está también disponible para otros lenguajes como Perl y Rubí.  
  
A pesar de su larga historia, su uso no está demasiado extendido entre los usuarios de equipos personales porque su integración visual con los sistemas operativos no era buena y proporcionaba pocos **widgets** (controles) para construir los programas gráficos.  
  
Sin embargo, a partir de **TKinter 8.5** la situación dio un giro de ciento ochenta grados en lo que se refiere a integración visual, mejorando en este aspecto notablemente; también en el número de widgets que se incluyen y en la posibilidad de trabajar con estilos y temas, que permiten ahora personalizar totalmente la estética de un programa. Por ello, ahora Tkinter es una alternativa atractiva y tan recomendable como otras.

**La primera aplicación con Tkinter**

El siguiente ejemplo crea una aplicación que incluye una ventana con un botón en la parte inferior. Al presionar el botón la aplicación termina su ejecución. Una ventana es el elemento fundamental de una aplicación GUI. Es el primer objeto que se crea y sobre éste se colocan el resto de objetos llamados widgets (etiquetas, botones, etc.).



#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

# Las dos líneas siguientes son necesaias para hacer

# compatible el interfaz Tkinter con los programas basados

# en versiones anteriores a la 8.5, con las más recientes.

from tkinter import \*    # Carga módulo tk (widgets estándar)

from tkinter import ttk  # Carga ttk (para widgets nuevos 8.5+)

# Define la ventana principal de la aplicación

raiz = Tk()

# Define las dimensiones de la ventana, que se ubicará en

# el centro de la pantalla. Si se omite esta línea la

# ventana se adaptará a los widgets que se coloquen en

# ella.

raiz.geometry('300x200') # anchura x altura

# Asigna un color de fondo a la ventana. Si se omite

# esta línea el fondo será gris

raiz.configure(bg = 'beige')

# Asigna un título a la ventana

raiz.title('Aplicación')

# Define un botón en la parte inferior de la ventana

# que cuando sea presionado hará que termine el programa.

# El primer parámetro indica el nombre de la ventana 'raiz'

# donde se ubicará el botón

ttk.Button(raiz, text='Salir', command=quit).pack(side=BOTTOM)

# Después de definir la ventana principal y un widget botón

# la siguiente línea hará que cuando se ejecute el programa

# construya y muestre la ventana, quedando a la espera de

# que alguna persona interactúe con ella.

# Si la persona presiona sobre el botón Cerrar 'X', o bien,

# sobre el botón 'Salir' el programa llegará a su fin.

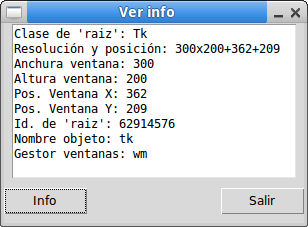
raiz.mainloop()

### La primera aplicación, orientada a objetos

A continuación, se muestra la misma aplicación pero [orientada a objetos](http://python-para-impacientes.blogspot.com.es/2014/02/programacion-orientada-objetos.html). Aunque este tipo de programación siempre es recomendable con Python no es imprescindible. Sin embargo, si vamos a trabajar con Tkinter es lo más adecuado, sobre todo, porque facilita la gestión de los widgets y de los eventos que se producen en las aplicaciones. Desde luego, todo van a ser ventajas.  
  
Normalmente, cuando se ejecuta una aplicación gráfica ésta se queda a la espera de que una persona interactúe con ella, que presione un botón, escriba algo en una caja de texto, seleccione una opción de un menú, sitúe el ratón en una posición determinada, etc., o bien, se produzca un suceso en el que no haya intervención humana como que termine un proceso, que cambie el valor de una variable, etc. En cualquiera de estos casos, lo habitual será vincular estos eventos o sucesos con unas acciones a realizar, que pueden ser mejor implementadas con las técnicas propias de la programación orientada a objetos.

**Obtener información de una ventana**

Para finalizar este capítulo se incluye una aplicación basada en los ejemplos anteriores que sirve para algo, concretamente, para mostrar información relacionada con la ventana.  
  
Para ello, en la ventana de la aplicación se han agregado nuevos widgets: un botón con la etiqueta "Info" y una caja de texto que aparece vacía.  
  
También, se ha incluido un método que será llamado cuando se presione el botón "Info" para obtener la información e insertarla en la caja de texto:



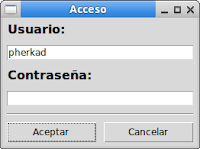
En la aplicación se utiliza el método **pack()** para ubicar los widgets en una posición determinada dentro de la ventana. Dicho método da nombre a uno de los tres **gestores de geometría** existentes en Tkinter, que son los responsables de esta tarea.

**Tkinter: Diseñando ventanas gráficas**

Para definir el modo en que deben colocarse los widgets (controles) dentro de una ventana se utilizan los gestores de geometría. En Tkinter existen tres gestores de geometría: **pack**, **grid** y **place**.  
  
Si una aplicación tiene varias ventanas, cada una de ellas puede estar construida con cualquiera de estos gestores, indistintamente. Será el desarrollador el que tendrá que elegir el que mejor resuelva el diseño que tenga por delante en cada momento.  
  
También, indicar que para construir las ventanas se pueden utilizar unos widgets especiales (marcos, paneles, etc.) que actúan como contenedores de otros widgets. Estos widgets se utilizan para agrupar varios controles al objeto de facilitar la operación a los usuarios. En las ventanas que se utilicen podrá emplearse un gestor con la ventana y otro diferente para organizar los controles dentro de estos widgets.  
  
A continuación, vamos a conocer las características de los tres gestores geométricos y a desarrollar una misma aplicación, utilizando cada uno de ellos.  
  
La aplicación consta de una ventana típica de acceso a un sistema que muestra en una caja de entrada la cuenta del usuario actual del equipo y presenta otra caja para introducir su contraseña. En la parte inferior hay dos botones: uno con el texto 'Aceptar' para validar la contraseña (mediante la llamada a un método) y otro con 'Cancelar' para finalizar la aplicación.

### El gestor de geometría Pack

Con este gestor la organización de los widgets se hace teniendo en cuenta los lados de una ventana: arriba, abajo, derecha e izquierda.  
  
Si varios controles se ubican (todos) en el lado de arriba o (todos) en el lado izquierdo de una ventana, construiremos una barra vertical o una barra horizontal de controles. Aunque es ideal para diseños simples (barras de herramientas, cuadros de diálogos, etc.) se puede utilizar también con diseños complejos. Además, es posible hacer que los controles se ajusten a los cambios de tamaño de la ventana.  
  
El ejemplo muestra la aplicación comentada con su ventana construida con el gestor **pack**:



Como hemos comentado antes la aplicación permite cambiar la dimensión de la ventana. Si lo hacemos los widgets se adaptarán al nuevo tamaño, teniendo en cuenta la configuración particular de cada uno de ellos. Para comprobar el funcionamiento podemos arrastrar los bordes de la ventana para ampliar o reducir el tamaño y comprobar como trabaja el gestor **pack**:

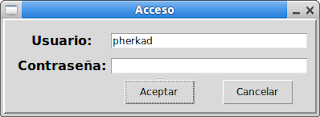
También, para verificar como actúa la opción **fill** podemos cambiar el valor actual **X** del widtget **self.ctext1.pack** por **Y** y arrastrar los bordes de la ventana. Si arrastramos hacia abajo el widget se expandirá verticalmente:

**El gestor de geometría Grid**

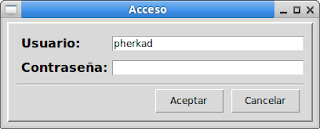
Este gestor geométrico trata una ventana como si fuera una cuadrícula, formada por filas y columnas como un tablero de ajedrez, donde es posible situar mediante una coordenada (fila, columna) los widgets; teniendo en cuenta que, si se requiere, un widget puede ocupar varias columnas y/o varias filas.  
  
Con este gestor es posible construir ventanas complejas y hacer que los controles se ajusten a un nuevo tamaño de las mismas. Se recomienda su uso con diseños en los que los controles deben aparecer alineados en varias columnas o filas, es decir, siguiendo la forma de una tabla.

### Grid con ventana no dimensionable

El siguiente ejemplo pretende ilustrar cómo usar el gestor **grid** con una ventana no dimensionable. También, utiliza un widget **Frame** con efecto 3D que contendrá al resto de controles:



**Grid con ventana dimensionable**



#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

from tkinter import \*

from tkinter import ttk, font

import getpass

# Gestor de geometría (grid). Ventana dimensionable

class Aplicacion():

    def \_\_init\_\_(self):

        self.raiz = Tk()

        self.raiz.title("Acceso")

        fuente = font.Font(weight='bold')

        self.marco = ttk.Frame(self.raiz, borderwidth=2,

                               relief="raised", padding=(10,10))

        self.etiq1 = ttk.Label(self.marco, text="Usuario:",

                               font=fuente, padding=(5,5))

        self.etiq2 = ttk.Label(self.marco, text="Contraseña:",

                               font=fuente, padding=(5,5))

        self.usuario = StringVar()

        self.clave = StringVar()

        self.usuario.set(getpass.getuser())

        self.ctext1 = ttk.Entry(self.marco, textvariable=self.usuario,

                                width=30)

        self.ctext2 = ttk.Entry(self.marco, textvariable=self.clave,

                                show="\*", width=30)

        self.separ1 = ttk.Separator(self.marco, orient=HORIZONTAL)

        self.boton1 = ttk.Button(self.marco, text="Aceptar",

                                 padding=(5,5), command=self.aceptar)

        self.boton2 = ttk.Button(self.marco, text="Cancelar",

                                 padding=(5,5), command=quit)

        # Para conseguir que la cuadricula y los widgets se

        # adapten al contenedor, si se amplia o reduce el tamaño

        # de la ventana, es necesario definir la opción 'sticky'.

        # Cuando un widget se ubica en el grid se coloca en el

        # centro de su celda o cuadro. Con 'sticky' se

        # establece el comportamiendo 'pegajoso' que tendrá el

        # widget dentro de su celda, cuando se modifique la

        # dimensión de la ventana. Para ello, se utilizan para

        # expresar sus valores los puntos cardinales: N (Norte),

        # S (Sur), (E) Este y (W) Oeste, que incluso se pueden

        # utilizar de forma combinada. El widget se quedará

        # 'pegado' a los lados de su celda en las direcciones

        # que se indiquen. cuando la ventana cambie de tamaño.

        # Pero con definir la opción 'sticky' no es suficiente:

        # hay activar esta propiedad más adelante.

        self.marco.grid(column=0, row=0, padx=5, pady=5,

                        sticky=(N, S, E, W))

        self.etiq1.grid(column=0, row=0,

                        sticky=(N, S, E, W))

        self.ctext1.grid(column=1, row=0, columnspan=2,

                         sticky=(E, W))

        self.etiq2.grid(column=0, row=1,

                        sticky=(N, S, E, W))

        self.ctext2.grid(column=1, row=1, columnspan=2,

                         sticky=(E, W))

        self.separ1.grid(column=0, row=3, columnspan=3, pady=5,

                         sticky=(N, S, E, W))

        self.boton1.grid(column=1, row=4, padx=5,

                         sticky=(E))

        self.boton2.grid(column=2, row=4, padx=5,

                         sticky=(W))

        # A continuación, se activa la propiedad de expandirse

        # o contraerse definida antes con la opción

        # 'sticky' del método grid().

        # La activación se hace por contenedores y por filas

        # y columnas asignando un peso a la opción 'weight'.

        # Esta opción asigna un peso (relativo) que se utiliza

        # para distribuir el espacio adicional entre columnas

        # y/o filas. Cuando se expanda la ventana, una columna

        # o fila con un peso 2 crecerá dos veces más rápido

        # que una columna (o fila) con peso 1. El valor

        # predeterminado es 0 que significa que la columna o

        # o fila no crecerá nada en absoluto.

        # Lo habitual es asignar pesos a filas o columnas donde

        # hay celdas con widgets.

        self.raiz.columnconfigure(0, weight=1)

        self.raiz.rowconfigure(0, weight=1)

        self.marco.columnconfigure(0, weight=1)

        self.marco.columnconfigure(1, weight=1)

        self.marco.columnconfigure(2, weight=1)

        self.marco.rowconfigure(0, weight=1)

        self.marco.rowconfigure(1, weight=1)

        self.marco.rowconfigure(4, weight=1)

        # Establece el foco en la caja de entrada de la

        # contraseña.

        self.ctext2.focus\_set()

        self.raiz.mainloop()

    def aceptar(self):

        if self.clave.get() == 'tkinter':

            print("Acceso permitido")

            print("Usuario:   ", self.ctext1.get())

            print("Contraseña:", self.ctext2.get())

        else:

            print("Acceso denegado")

            self.clave.set("")

            self.ctext2.focus\_set()

def main():

    mi\_app = Aplicacion()

    return 0

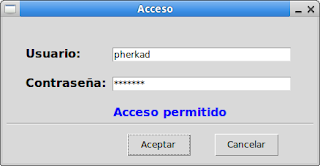
if \_\_name\_\_ == '\_\_main\_\_':

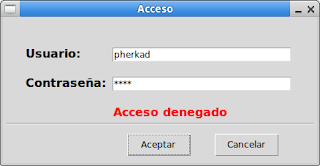
    main()

Después de ejecutar la aplicación, si ampliamos el tamaño de la ventana podemos comprobar como se ajustan los controles al nuevo espacio disponible según las direcciones descritas en cada opción **sticky**:

### El gestor de geometría Place

Este gestor es el más fácil de utilizar porque se basa en el posicionamiento absoluto para colocar los widgets, aunque el trabajo de "calcular" la posición de cada widget suele ser bastante laborioso. Sabemos que una ventana tiene una anchura y una altura determinadas (normalmente, medida en píxeles). Pues bien, con este método para colocar un widget simplemente tendremos que elegir la coordenada (x,y) de su ubicación expresada en píxeles.  
  
La posición (x=0, y=0) se encuentra en la esquina superior-izquierda de la ventana.  
  
Con este gestor el tamaño y la posición de un widget no cambiará al modificar las dimensiones de una ventana.  
  
Para finalizar, mostramos la famosa aplicación realizada con el gestor de geometría **place**. En este caso el modo de mostrar el mensaje de la validación se hace utilizando una etiqueta que cambia de color dependiendo si la contraseña es correcta o no. También, utiliza un método adicional para "limpiar" el mensaje de error cuando se haga clic con el ratón en la caja de entrada de la contraseña. El evento del widget se asocia con el método utilizando el método **bind()**.





#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

from tkinter import \*

from tkinter import ttk, font

import getpass

# Gestor de geometría (place)

class Aplicacion():

    def \_\_init\_\_(self):

        self.raiz = Tk()

        # Define la dimensión de la ventana

        self.raiz.geometry("430x200")

        # Establece que no se pueda cambiar el tamaño de la

        # ventana

        self.raiz.resizable(0,0)

        self.raiz.title("Acceso")

        self.fuente = font.Font(weight='bold')

        self.etiq1 = ttk.Label(self.raiz, text="Usuario:",

                               font=self.fuente)

        self.etiq2 = ttk.Label(self.raiz, text="Contraseña:",

                               font=self.fuente)

        # Declara una variable de cadena que se asigna a

        # la opción 'textvariable' de un widget 'Label' para

        # mostrar mensajes en la ventana. Se asigna el color

        # azul a la opción 'foreground' para el mensaje.

        self.mensa = StringVar()

        self.etiq3 = ttk.Label(self.raiz, textvariable=self.mensa,

                     font=self.fuente, foreground='blue')

        self.usuario = StringVar()

        self.clave = StringVar()

        self.usuario.set(getpass.getuser())

        self.ctext1 = ttk.Entry(self.raiz,

                                textvariable=self.usuario, width=30)

        self.ctext2 = ttk.Entry(self.raiz,

                                textvariable=self.clave,

                                width=30,

                                show="\*")

        self.separ1 = ttk.Separator(self.raiz, orient=HORIZONTAL)

        self.boton1 = ttk.Button(self.raiz, text="Aceptar",

                                 padding=(5,5), command=self.aceptar)

        self.boton2 = ttk.Button(self.raiz, text="Cancelar",

                                 padding=(5,5), command=quit)

        # Se definen las ubicaciones de los widgets en la

        # ventana asignando los valores de las opciones 'x' e 'y'

        # en píxeles.

        self.etiq1.place(x=30, y=40)

        self.etiq2.place(x=30, y=80)

        self.etiq3.place(x=150, y=120)

        self.ctext1.place(x=150, y=42)

        self.ctext2.place(x=150, y=82)

        self.separ1.place(x=5, y=145, bordermode=OUTSIDE,

                          height=10, width=420)

        self.boton1.place(x=170, y=160)

        self.boton2.place(x=290, y=160)

        self.ctext2.focus\_set()

        # El método 'bind()' asocia el evento de 'hacer clic

        # con el botón izquierdo del ratón en la caja de entrada

        # de la contraseña' expresado con '<button-1>' con el

        # método 'self.borrar\_mensa' que borra el mensaje y la

        # contraseña y devuelve el foco al mismo control.

        # Otros ejemplos de acciones que se pueden capturar:

        # <double-button-1>, <buttonrelease-1>, <enter>, <leave>,

        # <focusin>, <focusout>, <return>, <shift-up>, <key-f10>,

        # <key-space>, <key-print>, <keypress-h>, etc.

        self.raiz.mainloop()

    # Declara método para validar la contraseña y mostrar

    # un mensaje en la propia ventana, utilizando la etiqueta

    # 'self.mensa'. Cuando la contraseña es correcta se

    # asigna el color azul a la etiqueta 'self.etiq3' y

    # cuando es incorrecta el color rojo. Para ello. se emplea

    # el método 'configure()' que permite cambiar los valores

    # de las opciones de los widgets.

    def aceptar(self):

        if self.clave.get() == 'tkinter':

            self.etiq3.configure(foreground='blue')

            self.mensa.set("Acceso permitido")

        else:

            self.etiq3.configure(foreground='red')

            self.mensa.set("Acceso denegado")

    # Declara un método para borrar el mensaje anterior y

    # la caja de entrada de la contraseña

    def borrar\_mensa(self, evento):

        self.clave.set("")

        self.mensa.set("")

def main():

    mi\_app = Aplicacion()

    return 0

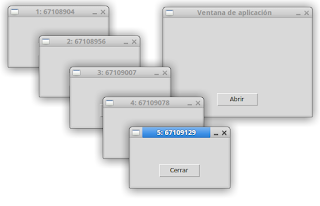
if \_\_name\_\_ == '\_\_main\_\_':

    main()

### Tkinter: Tipos de ventanas

**Ventanas de aplicación y de diálogos**

En la entrada anterior tratamos los distintos [gestores de geometría](http://python-para-impacientes.blogspot.com/2015/12/tkinter-disenando-ventanas-graficas.html) que se utilizan para diseñar las ventanas de una aplicación. A continuación, vamos a explorar los distintos tipos de ventanas que podemos crear y los usos que tienen.  
  
En **Tkinter** existen dos tipos de ventanas: las ***ventanas de aplicación***, que suelen ser las que inician y finalizan las aplicaciones gráficas; y desde las que se accede a las ***ventanas de diálogo***, que en conjunto constituyen la interfaz de usuario.  
  
Tanto unas como otras, desde el punto de vista del diseño, se construyen exactamente igual con los gestores de geometría ya conocidos: [**pack**, **grid** y **place**](http://python-para-impacientes.blogspot.com/2015/12/tkinter-disenando-ventanas-graficas.html).  
  
El siguiente ejemplo muestra una ventana de aplicación con un botón '**Abrir**' que cada vez que se presione abre una ventana hija (de diálogo) diferente y en una posición diferente. Observaremos que las ventanas hijas que se vayan generando se sitúan siempre en un plano superior con respecto a las creadas con anterioridad. Además, si abrimos varias ventanas podremos interactuar sin problemas con todas ellas, cambiar sus posiciones, cerrarlas, etc.  
  
En este caso, tanto para crear la ventana de aplicación como las hijas se utiliza el gestor de geometría **pack** pero notaremos algunas diferencias en el código:  
  
La ventana de aplicación se define con **Tk()**:  
  
**self.raiz = Tk()**  
  
Las ventanas de diálogo (hijas) se definen con **Toplevel()**:  
  
**self.dialogo = Toplevel()**  
  
El método **mainloop()** (bucle principal) hace que se atienda a los eventos de la ventana de aplicación:  
  
**self.raiz.mainloop()**  
  
El método **wait\_window()** hace que se atienda a los eventos locales de la ventana de diálogo mientras espera a ser cerrada:  
  
**self.raiz.wait\_window(self.conectar)**

[](https://4.bp.blogspot.com/-oqvQWjsdFV0/Vp1vpej2m3I/AAAAAAAACr4/ab0xYhMChBE/s1600/tkinter3-ventanas-no-modales.png)

#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

from tkinter import \*

from tkinter import ttk

class Aplicacion():

    ''' Clase Aplicacion '''

    # Declara una variable de clase para contar ventanas

    ventana = 0

    # Declara una variable de clase para usar en el

    # cálculo de la posición de una ventana

    posx\_y = 0

    def \_\_init\_\_(self):

        ''' Construye ventana de aplicación '''

        # Declara ventana de aplicación

        self.raiz = Tk()

        # Define dimensión de la ventana 300x200

        # que se situará en la coordenada x=500,y=50

        self.raiz.geometry('300x200+500+50')

        self.raiz.resizable(0,0)

        self.raiz.title("Ventana de aplicación")

        # Define botón 'Abrir' que se utilizará para

        # abrir las ventanas de diálogo. El botón

        # está vinculado con el método 'self.abrir'

        boton = ttk.Button(self.raiz, text='Abrir',

                           command=self.abrir)

        boton.pack(side=BOTTOM, padx=20, pady=20)

        self.raiz.mainloop()

    def abrir(self):

        ''' Construye una ventana de diálogo '''

        # Define una nueva ventana de diálogo

        self.dialogo = Toplevel()

        # Incrementa en 1 el contador de ventanas

        Aplicacion.ventana+=1

        # Recalcula posición de la ventana

        Aplicacion.posx\_y += 50

        tamypos = '200x100+'+str(Aplicacion.posx\_y)+ \

                  '+'+ str(Aplicacion.posx\_y)

        self.dialogo.geometry(tamypos)

        self.dialogo.resizable(0,0)

        # Obtiene identicador de la nueva ventana

        ident = self.dialogo.winfo\_id()

        # Construye mensaje de la barra de título

        titulo = str(Aplicacion.ventana)+": "+str(ident)

        self.dialogo.title(titulo)

        # Define el botón 'Cerrar' que cuando sea

        # presionado cerrará (destruirá) la ventana

        # 'self.dialogo' llamando al método

        # 'self.dialogo.destroy'

        boton = ttk.Button(self.dialogo, text='Cerrar',

                           command=self.dialogo.destroy)

        boton.pack(side=BOTTOM, padx=20, pady=20)

        # Cuando la ejecución del programa llega a este

        # punto se utiliza el método wait\_window() para

        # esperar que la ventana 'self.dialogo' sea

        # destruida.

        # Mientras tanto se atiende a los eventos locales

        # que se produzcan, por lo que otras partes de la

        # aplicación seguirán funcionando con normalidad.

        # Si hay código después de esta línea se ejecutará

        # cuando la ventana 'self.dialogo' sea cerrada.

        self.raiz.wait\_window(self.dialogo)

def main():

    mi\_app = Aplicacion()

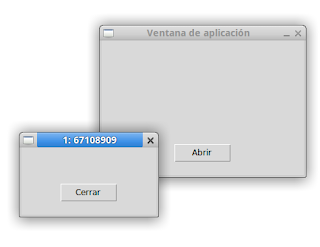
    return(0)

if \_\_name\_\_ == '\_\_main\_\_':

    main()

### Ventanas modales y no modales

Las ventanas hijas del ejemplo anterior son del tipo ***no modales*** porque mientras existen es posible interactuar libremente con ellas, sin ningún límite, excepto que si cerramos la ventana principal se cerrarán todas las ventanas hijas abiertas.  
  
Un ejemplo evidente que usa ventanas no modales está en las aplicaciones ofimáticas más conocidas, que permiten trabajar con varios documentos al mismo tiempo, cada uno de ellos abierto en su propia ventana, permitiendo al usuario cambiar sin restricciones de una ventana a otra.  
  
El caso contrario, es el de las **ventanas modales**. Cuando una ventana modal está abierta no será posible interactuar con otras ventanas de la aplicación hasta que ésta sea cerrada.  
  
Un ejemplo típico es el de algunas ventanas de diálogo que se utilizan para establecer las preferencias de las aplicaciones, que obligan a ser cerradas antes de permitirse la apertura de otras.  
  
Para demostrarlo, utilizamos el siguiente ejemplo en el que sólo es posible mantener abierta sólo una ventana hija, aunque si la cerramos podremos abrir otra.  
  
El método **grab\_set()** se utiliza para crear la ventana modal y el método **transiet()** se emplea para convertir la ventana de diálogo en ***ventana transitoria***, haciendo que se oculte cuando la ventana de aplicación sea minimizada.

[](https://1.bp.blogspot.com/-SxsoBp6Na58/Vp1y1xjsuQI/AAAAAAAACsE/pJYjQPNoy3s/s1600/tkinter3-ventanas-modales.png)

#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

from tkinter import \*

from tkinter import ttk

class Aplicacion():

    ventana = 0

    posx\_y = 0

    def \_\_init\_\_(self):

        self.raiz = Tk()

        self.raiz.geometry('300x200+500+50')

        self.raiz.resizable(0,0)

        self.raiz.title("Ventana de aplicación")

        boton = ttk.Button(self.raiz, text='Abrir',

                           command=self.abrir)

        boton.pack(side=BOTTOM, padx=20, pady=20)

        self.raiz.mainloop()

    def abrir(self):

        ''' Construye una ventana de diálogo '''

        self.dialogo = Toplevel()

        Aplicacion.ventana+=1

        Aplicacion.posx\_y += 50

        tamypos = '200x100+'+str(Aplicacion.posx\_y)+ \

                  '+'+ str(Aplicacion.posx\_y)

        self.dialogo.geometry(tamypos)

        self.dialogo.resizable(0,0)

        ident = self.dialogo.winfo\_id()

        titulo = str(Aplicacion.ventana)+": "+str(ident)

        self.dialogo.title(titulo)

        boton = ttk.Button(self.dialogo, text='Cerrar',

                           command=self.dialogo.destroy)

        boton.pack(side=BOTTOM, padx=20, pady=20)

        # Convierte la ventana 'self.dialogo' en

        # transitoria con respecto a su ventana maestra

        # 'self.raiz'.

        # Una ventana transitoria siempre se dibuja sobre

        # su maestra y se ocultará cuando la maestra sea

        # minimizada. Si el argumento 'master' es

        # omitido el valor, por defecto, será la ventana

        # madre.

        self.dialogo.transient(master=self.raiz)

        # El método grab\_set() asegura que no haya eventos

        # de ratón o teclado que se envíen a otra ventana

        # diferente a 'self.dialogo'. Se utiliza para

        # crear una ventana de tipo modal que será

        # necesario cerrar para poder trabajar con otra

        # diferente. Con ello, también se impide que la

        # misma ventana se abra varias veces.

        self.dialogo.grab\_set()

        self.raiz.wait\_window(self.dialogo)

def main():

    mi\_app = Aplicacion()

    return(0)

if \_\_name\_\_ == '\_\_main\_\_':

    main()

### Variables de control

Las variables de control son objetos especiales que se asocian a los widgets para almacenar sus valores y facilitar su disponibilidad en otras partes del programa. Pueden ser de tipo numérico, de cadena y booleano.  
  
Cuando una variable de control cambia de valor el widget que la utiliza lo refleja automáticamente, y viceversa.  
  
Las variables de control también se emplean para conectar varios widgets del mismo tipo, por ejemplo, varios controles del tipo **Radiobutton**. En este caso tomarán un valor de varios posibles.

### Declarar variables de control

Las variables de control se declaran de forma diferente en función al tipo de dato que almacenan:

entero = IntVar() # Declara variable de tipo entera

flotante = DoubleVar() # Declara variable de tipo flotante

cadena = StringVar() # Declara variable de tipo cadena

booleano = BooleanVar() # Declara variable de tipo booleana

También, en el momento de declarar una variable es posible asignar un valor inicial:

blog = StringVar(value="Python para impacientes")

### Método set()

El método **set()** asigna un valor a una variable de control. Se utiliza para modificar el valor o estado de un widget:

nombre = StringVar()

id\_art = IntVar()

nombre.set("Python para impacientes")

id\_art.set(1)

blog = ttk.Entry(ventana, textvariable=nombre, width=25)

arti = ttk.Label(ventana, textvariable=id\_art)

### Método get()

El método **get()** obtiene el valor que tenga, en un momento dado, una variable de control. Se utiliza cuando es necesario leer el valor de un control:

print('Blog:', nombre.get())

print('Id artículo:', id\_art.get())

### Método trace()

El método **trace()** se emplea para "detectar" cuando una variable es leída, cambia de valor o es borrada:  
  
**widget.trace(tipo, función)**  
  
El primer argumento establece el tipo de suceso a comprobar: '**r**' lectura de variable, '**w**' escritura de variable y '**u**' borrado de variable. El segundo argumento indica la función que será llamada cuando se produzca el suceso.  
  
En el siguiente ejemplo se define una variable de control de tipo cadena y con el método **trace()** se asocian su lectura y cambio de valor a dos funciones que son llamadas cuando ocurran estos sucesos. Concretamente, cuando se utilice el método **set()** se llamará a la función '**cambia()**' y cuando se use **get()** a la función '**lee()**'.

def cambia(\*args):

print("Ha cambiado su valor")

def lee(\*args):

print("Ha sido leido su valor")

variable = StringVar()

variable.trace("w", cambia)

variable.trace("r", lee)

variable.set("Hola")

print(variable.get())

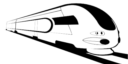
print(variable.get())

### Estrategias para validar y calcular datos

Cuando se construye una ventana con varios widgets se pueden seguir distintas estrategias para validar los datos que se introducen durante la ejecución de un programa:

* Una opción posible podría validar la información y realizar los cálculos después de que sea introducida, por ejemplo, después de presionar un botón.
* Otra posibilidad podría ser haciendo uso del método **trace()** y de la opción '**command**', para validar y calcular la información justo en el momento que un widget y su variable asociada cambien de valor.

A continuación, se muestra un mismo caso práctico utilizando ambas técnicas.

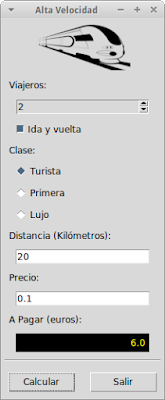
[](https://3.bp.blogspot.com/-Onau2Texukk/VrHzeMqc5oI/AAAAAAAACss/dwYvBmRm0_o/s1600/tren-128x64.png)

El programa de este ejemplo calcula el coste de un viaje en tren teniendo en cuenta el número de viajeros; el tipo de billete (de sólo ida o de ida y vuelta); la clase en la cual se viaja (que puede ser clase turista, primera o lujo); la distancia en kilómetros y el precio por kilómetro (por defecto es 0,10 céntimos de euro).  
  
El cálculo del importe a pagar se realiza multiplicando número de viajeros por km y precio, con los siguientes incrementos:

* Si el viaje es de ida y vuelta se multiplica el total por 1,5
* Si la clase es primera se multiplica el total por 1,2 y si es de lujo se multiplica por 2

### Validación y cálculo posterior

En la primera solución la validación de los datos y el cálculo del importe a pagar se realiza después presionar el botón "Calcular".



#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

from tkinter import \*

from tkinter import ttk

# Calcula coste de viaje con validación y cálculo posterior

class Aplicacion():

    def \_\_init\_\_(self):

        self.raiz = Tk()

        self.raiz.title("Alta Velocidad")

        # Declara variables de control

        self.num\_via = IntVar(value=1)

        self.ida\_vue = BooleanVar()

        self.clase   = StringVar(value='t')

        self.km = IntVar(value=1)

        self.precio = DoubleVar(value=0.10)

        self.total = DoubleVar(value=0.0)

        # Carga imagen para asociar a widget Label()

        tren = PhotoImage(file='tren-128x64.png')

        # Declara widgets de la ventana

        # Se incluye el widget de tipo Button 'Calcular' que utiliza

        # la opción 'command' para validar datos y calcular el

        # importe a pagar cuando sea presionado

        self.imagen1 = ttk.Label(self.raiz, image=tren,

                                 anchor="center")

        self.etiq1 = ttk.Label(self.raiz, text="Viajeros:")

        self.viaje = Spinbox(self.raiz, from\_=1, to=20, wrap=True,

                             textvariable=self.num\_via,

                             state='readonly')

        self.idavue = ttk.Checkbutton(self.raiz, text='Ida y vuelta',

                                      variable=self.ida\_vue,

                                      onvalue=True, offvalue=False)

        self.etiq2 = ttk.Label(self.raiz, text="Clase:")

        self.clase1 = ttk.Radiobutton(self.raiz, text='Turista',

                                      variable=self.clase, value='t')

        self.clase2 = ttk.Radiobutton(self.raiz, text='Primera',

                                      variable=self.clase, value='p')

        self.clase3 = ttk.Radiobutton(self.raiz, text='Lujo',

                                       variable=self.clase, value='l')

        self.etiq3 = ttk.Label(self.raiz,

                               text="Distancia Kilómetros):")

        self.dist = ttk.Entry(self.raiz, textvariable=self.km,

                              width=10)

        self.etiq4 = ttk.Label(self.raiz, text="Precio:")

        self.coste = ttk.Entry(self.raiz, textvariable=self.precio,

                               width=10)

        self.etiq5 = ttk.Label(self.raiz, text="A Pagar (euros):")

        self.etiq6 = ttk.Label(self.raiz, textvariable=self.total,

                               foreground="yellow", background="black",

                               borderwidth=5, anchor="e")

        self.separ1 = ttk.Separator(self.raiz, orient=HORIZONTAL)

        self.boton1 = ttk.Button(self.raiz, text="Calcular",

                                 command=self.calcular)

        self.boton2 = ttk.Button(self.raiz, text="Salir",

                                 command=quit)

        self.imagen1.pack(side=TOP, fill=BOTH, expand=True,

                          padx=10, pady=5)

        self.etiq1.pack(side=TOP, fill=BOTH, expand=True,

                        padx=10, pady=5)

        self.viaje.pack(side=TOP, fill=X, expand=True,

                        padx=20, pady=5)

        self.idavue.pack(side=TOP, fill=X, expand=True,

                         padx=20, pady=5)

        self.etiq2.pack(side=TOP, fill=BOTH, expand=True,

                        padx=10, pady=5)

        self.clase1.pack(side=TOP, fill=BOTH, expand=True,

                         padx=20, pady=5)

        self.clase2.pack(side=TOP, fill=BOTH, expand=True,

                         padx=20, pady=5)

        self.clase3.pack(side=TOP, fill=BOTH, expand=True,

                         padx=20, pady=5)

        self.etiq3.pack(side=TOP, fill=BOTH, expand=True,

                        padx=10, pady=5)

        self.dist.pack(side=TOP, fill=X, expand=True,

                       padx=20, pady=5)

        self.etiq4.pack(side=TOP, fill=BOTH, expand=True,

                        padx=10, pady=5)

        self.coste.pack(side=TOP, fill=X, expand=True,

                        padx=20, pady=5)

        self.etiq5.pack(side=TOP, fill=BOTH, expand=True,

                        padx=10, pady=5)

        self.etiq6.pack(side=TOP, fill=BOTH, expand=True,

                        padx=20, pady=5)

        self.separ1.pack(side=TOP, fill=BOTH, expand=True,

                         padx=5, pady=5)

        self.boton1.pack(side=LEFT, fill=BOTH, expand=True,

                         padx=10, pady=10)

        self.boton2.pack(side=RIGHT, fill=BOTH, expand=True,

                         padx=10, pady=10)

        self.raiz.mainloop()

    def calcular(self):

        # Función para validar datos y calcular importe a pagar

        error\_dato = False

        total = 0

        try:

            km = int(self.km.get())

            precio = float(self.precio.get())

        except:

            error\_dato = True

        if not error\_dato:

            total =  self.num\_via.get() \* km \* precio

            if self.ida\_vue.get():

                total = total \* 1.5

            if self.clase.get() == 'p':

                total = total \* 1.2

            elif self.clase.get() == 'l':

                total = total \* 2

            self.total.set(total)

        else:

            self.total.set("¡ERROR!")

def main():

    mi\_app = Aplicacion()

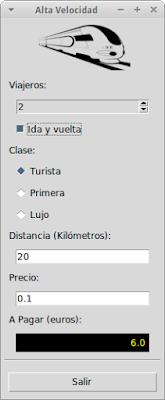
    return 0

if \_\_name\_\_ == '\_\_main\_\_':

    main()

### Validación y cálculo inmediato

En la segunda solución no se utiliza el botón '**Calcular**'. La validación y el cálculo se realizan al cambiar el valor de cualquier widget, mostrando el resultado inmediatamente. Para los widget de entrada de datos (**Entry()**) se definen dos trazas para detectar cualquier cambio en los datos y en el resto de widgets se utiliza la opción '**command**'.



#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

from tkinter import \*

from tkinter import ttk

# Calcula coste de viaje con validación y cálculo inmediato

class Aplicacion():

    def \_\_init\_\_(self):

        self.raiz = Tk()

        self.raiz.title("Alta Velocidad")

        # Declara variables de control

        self.num\_via = IntVar(value=1)

        self.ida\_vue = BooleanVar()

        self.clase   = StringVar(value='t')

        self.km = IntVar(value=1)

        self.precio = DoubleVar(value=0.10)

        self.total = DoubleVar(value=0.0)

        # Define trazas con variables de control de los widgets Entry()

        # para detectar cambios en los datos. Si se producen cambios

        # se llama a la función 'self.calcular' para validación y para

        # calcular importe a pagar

        self.km.trace('w', self.calcular)

        self.precio.trace('w', self.calcular)

        # Llama a función para validar y calcular

        self.calcular()

        # Carga imagen para asociar a widget Label()

        tren = PhotoImage(file='tren-128x64.png')

        # Declara widgets de la ventana

        # En los widgets de tipo Spinbox, Checkbutton y Radiobutton

        # se utiliza la opción 'command' para llamar a la función

        # 'self.calcular' para validar datos y calcular importe a

        # pagar de forma inmediata

        self.imagen1 = ttk.Label(self.raiz, image=tren,

                                 anchor="center")

        self.etiq1 = ttk.Label(self.raiz, text="Viajeros:")

        self.viaje = Spinbox(self.raiz, from\_=1, to=20, wrap=True,

                             textvariable=self.num\_via,

                             state='readonly',

                             command=self.calcular)

        self.idavue = ttk.Checkbutton(self.raiz, text='Ida y vuelta',

                                      variable=self.ida\_vue,

                                      onvalue=True, offvalue=False,

                                      command=self.calcular)

        self.etiq2 = ttk.Label(self.raiz, text="Clase:")

        self.clase1 = ttk.Radiobutton(self.raiz, text='Turista',

                                      variable=self.clase, value='t',

                                      command=self.calcular)

        self.clase2 = ttk.Radiobutton(self.raiz, text='Primera',

                                      variable=self.clase, value='p',

                                      command=self.calcular)

        self.clase3 = ttk.Radiobutton(self.raiz, text='Lujo',

                                      variable=self.clase, value='l',

                                      command=self.calcular)

        self.etiq3 = ttk.Label(self.raiz,

                               text="Distancia (Kilómetros):")

        self.dist = ttk.Entry(self.raiz, textvariable=self.km,

                              width=10)

        self.etiq4 = ttk.Label(self.raiz, text="Precio:")

        self.coste = ttk.Entry(self.raiz, textvariable=self.precio,

                               width=10)

        self.etiq5 = ttk.Label(self.raiz, text="A Pagar (euros):")

        self.etiq6 = ttk.Label(self.raiz, textvariable=self.total,

                               foreground="yellow", background="black",

                               borderwidth=5, anchor="e")

        self.separ1 = ttk.Separator(self.raiz, orient=HORIZONTAL)

        self.boton1 = ttk.Button(self.raiz, text="Salir",

                                 command=quit)

        self.imagen1.pack(side=TOP, fill=BOTH, expand=True,

                          padx=10, pady=5)

        self.etiq1.pack(side=TOP, fill=BOTH, expand=True,

                        padx=10, pady=5)

        self.viaje.pack(side=TOP, fill=X, expand=True,

                        padx=20, pady=5)

        self.idavue.pack(side=TOP, fill=X, expand=True,

                         padx=20, pady=5)

        self.etiq2.pack(side=TOP, fill=BOTH, expand=True,

                        padx=10, pady=5)

        self.clase1.pack(side=TOP, fill=BOTH, expand=True,

                         padx=20, pady=5)

        self.clase2.pack(side=TOP, fill=BOTH, expand=True,

                         padx=20, pady=5)

        self.clase3.pack(side=TOP, fill=BOTH, expand=True,

                         padx=20, pady=5)

        self.etiq3.pack(side=TOP, fill=BOTH, expand=True,

                        padx=10, pady=5)

        self.dist.pack(side=TOP, fill=X, expand=True,

                       padx=20, pady=5)

        self.etiq4.pack(side=TOP, fill=BOTH, expand=True,

                        padx=10, pady=5)

        self.coste.pack(side=TOP, fill=X, expand=True,

                        padx=20, pady=5)

        self.etiq5.pack(side=TOP, fill=BOTH, expand=True,

                        padx=10, pady=5)

        self.etiq6.pack(side=TOP, fill=BOTH, expand=True,

                        padx=20, pady=5)

        self.separ1.pack(side=TOP, fill=BOTH, expand=True,

                         padx=5, pady=5)

        self.boton1.pack(side=RIGHT, fill=BOTH, expand=True,

                         padx=10, pady=10)

        self.raiz.mainloop()

    def calcular(self, \*args):

        # Función para validar datos y calcular importe a pagar

        error\_dato = False

        total = 0

        try:

            km = int(self.km.get())

            precio = float(self.precio.get())

        except:

            error\_dato = True

        if not error\_dato:

            total =  self.num\_via.get() \* km \* precio

            if self.ida\_vue.get():

                total = total \* 1.5

            if self.clase.get() == 'p':

                total = total \* 1.2

            elif self.clase.get() == 'l':

                total = total \* 2

            self.total.set(total)

        else:

            self.total.set("¡ERROR!")

def main():

    mi\_app = Aplicacion()

    return 0

if \_\_name\_\_ == '\_\_main\_\_':

    main()

**Menús, barras de herramientas y de estado en Tkinter**

**Introducción**

Tkinter cuenta con widgets específicos para incluir **menús de opciones** de distintos tipos en una aplicación. Además, siguiendo unas pautas en la construcción de ventanas es posible agregar otros elementos como **barras de herramientas y de estado**.  
  
Todos estos componentes se utilizan con frecuencia en el desarrollo de interfaces porque facilitan mucho la interacción de los usuarios con las aplicaciones gráficas.

**Menús de opciones**

Los menús pueden construirse agrupando en una barra de menú varios submenús desplegables, mediante menús basados en botones, o bien, utilizando los típicos menús contextuales que cambian sus opciones disponibles dependiendo del lugar donde se activan en la ventana de la aplicación.  
  
Entre los tipos de opciones que se pueden incluir en un menú se encuentran aquellas que su estado representan un valor lógico de activada o desactivada (**add\_checkbutton**); las que permiten elegir una opción de entre varias existentes (**add\_radiobutton**) y las que ejecutan directamente un método o una función (**add\_command**).  
  
Las opciones de un menú pueden incluir iconos y asociarse a atajos o combinaciones de teclas que surten el mismo efecto que si éstas son seleccionadas con un clic de ratón. También, en un momento dado, pueden deshabilitarse para impedir que puedan ser seleccionadas.

**Barras de herramientas**

Una aplicación Tkinter además de menús de opciones puede incorporar barras de herramientas construidas a base de botones apilados de manera horizontal o vertical. Estas barras de herramientas suelen situarse, muchas veces, justo debajo de la barra de menú de la aplicación y ejecutan los procesos más utilizados en el sistema.

**Barra de estado**

Las barras de estado se utilizan con asiduidad en los programas gráficos para mostrar información que resulte útil a los usuarios. Normalmente, ocupan la última línea de la ventana de aplicación y en algunos casos puede ocultarse para dejar más espacio disponible a dicha ventana.

**PyRemoto, un ejemplo de implementación**

**PyRemoto** es un ejemplo que pretende mostrar cómo incluir menús y barras en una aplicación basada en Tkinter.  
  
La ventana principal de esta aplicación cuenta con una barra de título, una barra de menú con tres submenús desplegables con distintos tipos de opciones, un menú contextual que se activa haciendo clic con el botón secundario del ratón en cualquier lugar de la ventana, una barra de herramientas con dos botones con imágenes y una barra de estado que es posible ocultar o mostrar mediante la opción "**Ver barra de estado**" del submenú "**Opciones**".

En el submenú "**Opciones**" se encuentra la opción "**Guardar**" que se habilitará cuando se elija alguna opción de entre las disponibles en este submenú; y se deshabilitará cuando se utilice la propia opción "**Guardar**", -prevista- para salvar las preferencias seleccionadas por el usuario

[[https://1.bp.blogspot.com/-z--3pxcvPWE/VvZ6UKLQ6oI/AAAAAAAACuQ/QLlAT5-k9bUVViuGo5ZKMYgEjnjgZT_Dw/s1600/conec16x16.png](https://1.bp.blogspot.com/-z--3pxcvPWE/VvZ6UKLQ6oI/AAAAAAAACuQ/QLlAT5-k9bUVViuGo5ZKMYgEjnjgZT_Dw/s1600/conec16x16.png)](https://1.bp.blogspot.com/-z--3pxcvPWE/VvZ6UKLQ6oI/AAAAAAAACuQ/QLlAT5-k9bUVViuGo5ZKMYgEjnjgZT_Dw/s1600/conec16x16.png)

[https://4.bp.blogspot.com/-VdXU1wJEbnU/VvZ6UGKbb2I/AAAAAAAACuA/quK-vvVjVCMliVAKVB_s8BoC-3DGvoNgA/s1600/conec32x32.png](https://4.bp.blogspot.com/-VdXU1wJEbnU/VvZ6UGKbb2I/AAAAAAAACuA/quK-vvVjVCMliVAKVB_s8BoC-3DGvoNgA/s1600/conec32x32.png)

[](https://4.bp.blogspot.com/-VWJ0fVvjx0Y/VvZ6UCY70tI/AAAAAAAACt8/_xbhyI6BBiwlLTVN51VlMUx9bqjxIVUPQ/s1600/pyremoto64x64.png)

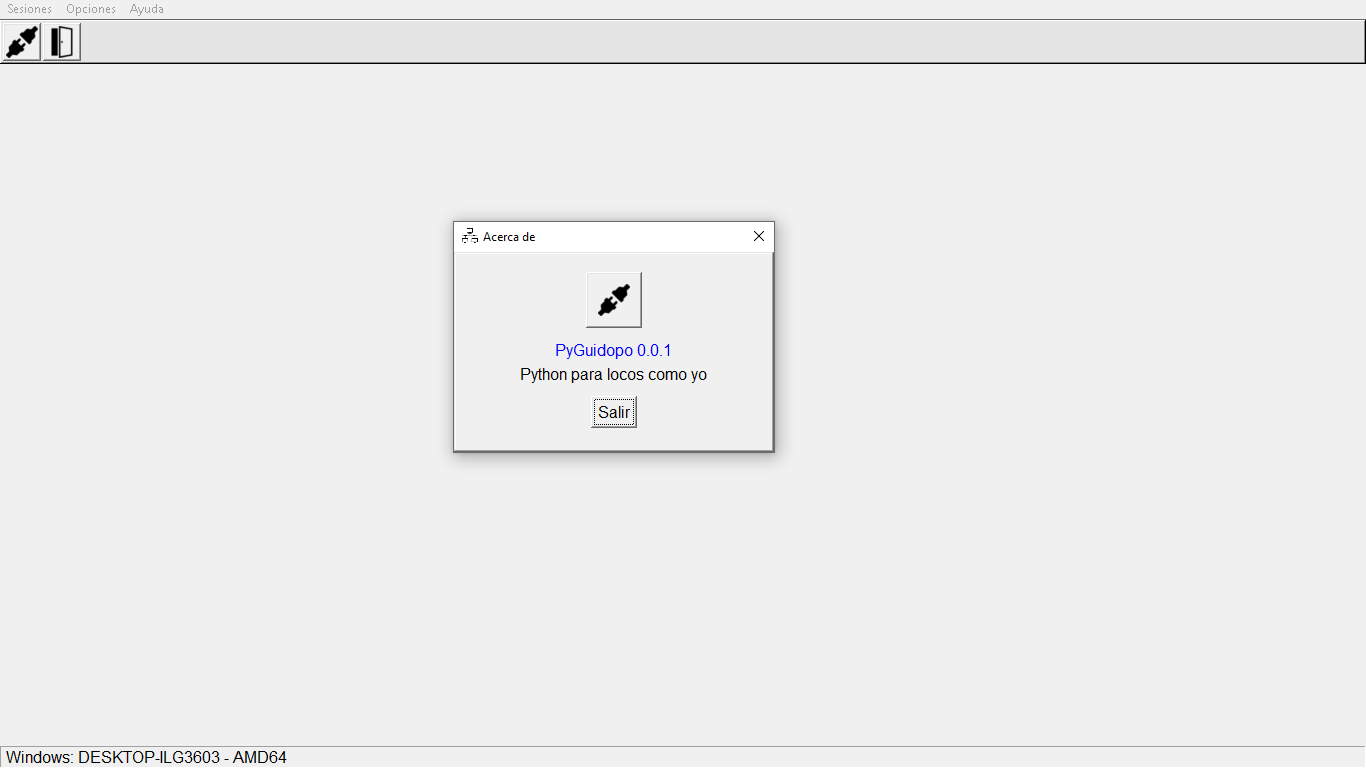
[https://2.bp.blogspot.com/-0oNTNXbrtZg/VvZ6UXaHnkI/AAAAAAAACuE/4EkHNx-fTVkq2llDMUzmnpT6npUn4KOsw/s1600/salir16x16.png](https://2.bp.blogspot.com/-0oNTNXbrtZg/VvZ6UXaHnkI/AAAAAAAACuE/4EkHNx-fTVkq2llDMUzmnpT6npUn4KOsw/s1600/salir16x16.png)

[https://4.bp.blogspot.com/-OC-IVi5rdQI/VvZ6UssnXTI/AAAAAAAACuI/DEjgLofSSvQ7-cGZEdOx5Xi5qw2WFEABg/s1600/salir32x32.png](https://4.bp.blogspot.com/-OC-IVi5rdQI/VvZ6UssnXTI/AAAAAAAACuI/DEjgLofSSvQ7-cGZEdOx5Xi5qw2WFEABg/s1600/salir32x32.png)

[https://2.bp.blogspot.com/-lkewQRkpoUw/VvZ6VAtToUI/AAAAAAAACuM/Fi92c6_3eCQ5U8s8hM2rE2L-SlwaeY4HA/s1600/star16x16.png](https://2.bp.blogspot.com/-lkewQRkpoUw/VvZ6VAtToUI/AAAAAAAACuM/Fi92c6_3eCQ5U8s8hM2rE2L-SlwaeY4HA/s1600/star16x16.png)

La aplicación de ejemplo utiliza imágenes de diferentes tamaños que se localizan en una carpeta llamada "***imagen***". Estas imágenes son utilizadas en la barra de título, los submenús, la barra de herramientas y en la ventana de diálogo "**Acerca de**" del submenú "**Ayuda**". Han sido obtenidas del sitio [flaticon.com](http://www.flaticon.com/) que ofrece colecciones de imágenes organizadas por distintas temáticas a programadores y a diseñadores gráficos.

También, en el programa se incluyen varias combinaciones de teclas asociadas a distintas opciones del menú, declaradas con el método **bind()**.  
  
  
**¿Cómo funciona?**  
  
La aplicación la inicia la función **main()** que comprueba que todas las imágenes de la aplicación existen en la carpeta "***imagen***". Si todas las imágenes están disponibles se creará el objeto aplicación mediante la clase **PyRemoto()**. Esta clase cuenta con un método **\_\_init\_\_()** que construye la ventana de la aplicación, los menús, la barra de herramientas y una barra de estado.  
  
La aplicación está incompleta pero enseña cómo organizar los métodos que son llamados desde las distintas opciones de los menús y barras.  
  
Por último, el ejemplo incluye la ventana de diálogo "**Acerca de**", que es de tipo modal, para mostrar el modo de abrir este tipo de ventanas desde una opción del menú.



#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

#

# name:        pyremoto.py (Python 3.x).

# description: Acceso Remoto a equipos por ssh, sftp y rdp

# purpose:     Construcción de menús, barras de herramientas

#              y de estado

# author:      python para impacientes

#

#------------------------------------------------------------

'''PyRemoto: Acceso Remoto a equipos por ssh, sftp y rdp '''

\_\_author\_\_ = 'python para impacientes'

\_\_title\_\_= 'PyRemoto'

\_\_date\_\_ = ''

\_\_version\_\_ = '0.0.1'

\_\_license\_\_ = 'GNU GPLv3'

import os, sys, webbrowser, platform

from tkinter import \*

from tkinter import ttk, font, messagebox

# PYREMOTO: ACCESO REMOTO A EQUIPOS POR SSH, SFTP y RDP

class PyRemoto():

    ''' Clase PyRemoto '''

    # DECLARAR MÉTODO CONSTRUCTOR DE LA APLICACIÓN

    def \_\_init\_\_(self, img\_carpeta, iconos):

        ''' Definir ventana de la aplicación, menú, submenús,

            menú contextual, barra de herramientas, barra de

            estado y atajos del teclado '''

        # INICIALIZAR VARIABLES

        self.img\_carpeta = img\_carpeta

        self.iconos = iconos

        # DEFINIR VENTANA DE LA APLICACIÓN:

        self.raiz = Tk()

        # ESTABLECER PROPIEDADES DE LA VENTANA DE APLICACIÓN:

        self.raiz.title("PyRemoto " + \_\_version\_\_)  # Título

        self.icono1= PhotoImage(file=self.iconos[0])  # Icono app

        self.raiz.iconphoto(self.raiz, self.icono1)  # Asigna icono app

        self.raiz.option\_add("\*Font", "Helvetica 12")  # Fuente predeterminada

        self.raiz.option\_add('\*tearOff', False)  # Deshabilita submenús flotantes

        self.raiz.attributes('-fullscreen', True)  # Maximiza ventana completa

        self.raiz.minsize(400,300)  # Establece tamaño minimo ventana

        # ESTABLECER ESTILO FUENTE PARA ALGUNOS WIDGETS:

        self.fuente = font.Font(weight='normal') # normal, bold, etc...

        # DECLARAR VARIABLES PARA OPCIONES PREDETERMINADAS:

        # (Estos valores se podrían leer de un archivo de

        # configuración)

        self.CFG\_TIPOCONEX = IntVar()

        self.CFG\_TIPOCONEX.set(1) # shh

        self.CFG\_TIPOEMUT = IntVar()

        self.CFG\_TIPOEMUT.set(1) # xterm

        self.CFG\_TIPOEXP = IntVar()

        self.CFG\_TIPOEXP.set(1) # thunar

        # DECLARAR VARIABLE PARA MOSTRAR BARRA DE ESTADO:

        self.estado = IntVar()

        self.estado.set(1)  # Mostrar Barra de Estado

        # DEFINIR BARRA DE MENÚ DE LA APLICACION:

        barramenu = Menu(self.raiz)

        self.raiz['menu'] = barramenu

        # DEFINIR SUBMENÚS 'Sesiones', 'Opciones' y 'Ayuda':

        menu1 = Menu(barramenu)

        self.menu2 = Menu(barramenu)

        menu3 = Menu(barramenu)

        barramenu.add\_cascade(menu=menu1, label='Sesiones')

        barramenu.add\_cascade(menu=self.menu2, label='Opciones')

        barramenu.add\_cascade(menu=menu3, label='Ayuda')

        # DEFINIR SUBMENÚ 'Sesiones':

        icono2 = PhotoImage(file=self.iconos[1])

        icono3 = PhotoImage(file=self.iconos[2])

        menu1.add\_command(label='Conectar...',

                          command=self.f\_conectar,

                          underline=0, accelerator="Ctrl+c",

                          image=icono2, compound=LEFT)

        menu1.add\_separator()  # Agrega un separador

        menu1.add\_command(label='Salir', command=self.f\_salir,

                          underline=0, accelerator="Ctrl+q",

                          image=icono3, compound=LEFT)

        # DEFINIR SUBMENÚ 'Opciones':

        self.menu2.add\_checkbutton(label='Barra de Estado',

                          variable=self.estado, onvalue=1,

                          offvalue=0,

                          command=self.f\_verestado)

        self.menu2.add\_separator()

        self.menu2.add\_radiobutton(label='ssh',

                          variable=self.CFG\_TIPOCONEX,

                          command=self.f\_cambiaropc,

                          value=1)

        self.menu2.add\_radiobutton(label='sftp',

                          variable=self.CFG\_TIPOCONEX,

                          command=self.f\_cambiaropc,

                          value=2)

        self.menu2.add\_radiobutton(label='rdp',

                          variable=self.CFG\_TIPOCONEX,

                          command=self.f\_cambiaropc,

                          value=3)

        self.menu2.add\_separator()

        self.menu2.add\_radiobutton(label='xterm',

                          variable=self.CFG\_TIPOEMUT,

                          command=self.f\_cambiaropc,

                          value=1)

        self.menu2.add\_radiobutton(label='uxterm',

                          variable=self.CFG\_TIPOEMUT,

                          command=self.f\_cambiaropc,

                          value=2)

        self.menu2.add\_radiobutton(label='xfce4-terminal',

                          variable=self.CFG\_TIPOEMUT,

                          command=self.f\_cambiaropc,

                          value=3)

        self.menu2.add\_separator()

        self.menu2.add\_radiobutton(label='Thunar',

                          variable=self.CFG\_TIPOEXP,

                          command=self.f\_cambiaropc,

                          value=1)

        self.menu2.add\_radiobutton(label='Nautilus',

                          variable=self.CFG\_TIPOEXP,

                          command=self.f\_cambiaropc,

                          value=2)

        self.menu2.add\_separator()

        self.menu2.add\_command(label='Guardar',

                          command=self.f\_opcionguardar,

                          state="disabled", underline=0,

                          accelerator="Ctrl+g")

        # DEFINIR SUBMENÚ 'Ayuda':

        menu3.add\_command(label='Web', command=self.f\_web)

        menu3.add\_command(label='Atajos teclado',

                          command=self.f\_atajos)

        icono4 = PhotoImage(file=self.iconos[3])

        menu3.add\_command(label="Acerca de",

                          command=self.f\_acerca,

                          image=icono4, compound=LEFT)

        # DEFINIR BARRA DE HERRAMIENTAS:

        self.icono5 = PhotoImage(file=self.iconos[4])

        icono6 = PhotoImage(file=self.iconos[5])

        barraherr = Frame(self.raiz, relief=RAISED,

                          bd=2, bg="#E5E5E5")

        bot1 = Button(barraherr, image=self.icono5,

                      command=self.f\_conectar)

        bot1.pack(side=LEFT, padx=1, pady=1)

        bot2 = Button(barraherr, image=icono6,

                      command=self.f\_salir)

        bot2.pack(side=LEFT, padx=1, pady=1)

        barraherr.pack(side=TOP, fill=X)

        # DEFINIR BARRA DE ESTADO:

        # Muestra información del equipo

        info1 = platform.system()

        info2 = platform.node()

        info3 = platform.machine()

        # Otro modo de obtener la información:

        # (No disponible en algunas versiones de Windows)

        #info1 = os.uname().sysname

        #info2 = os.uname().nodename

        #info3 = os.uname().machine

        mensaje = " " + info1+ ": "+info2+" - "+info3

        self.barraest = Label(self.raiz, text=mensaje,

                              bd=1, relief=SUNKEN, anchor=W)

        self.barraest.pack(side=BOTTOM, fill=X)

        # DEFINIR MENU CONTEXTUAL

        self.menucontext = Menu(self.raiz, tearoff=0)

        self.menucontext.add\_command(label="Conectar",

                                     command=self.f\_conectar)

        self.menucontext.add\_command(label="Salir",

                                     command=self.f\_salir)

        # DECLARAR TECLAS DE ACCESO RAPIDO:

        self.raiz.bind("<Control-c>",

                       lambda event: self.f\_conectar())

        self.raiz.bind("<Control-g>",

                       lambda event: self.f\_guardar())

        self.raiz.bind("<Control-q>",

                       lambda event: self.f\_salir())

        self.raiz.bind("<Button-3>",

                       self.f\_mostrarmenucontext)

        self.raiz.mainloop()

    # DECLARAR OTROS MÉTODOS DE LA APLICACIÓN:

    def f\_opcionguardar(self):

        ''' Si opción 'Guardar' está habilitada llama a

            método para guardar opciones de configuración

            de la aplicación '''

        if self.menu2.entrycget(13,"state")=="normal":

            self.menu2.entryconfig(13, state="disabled")

            self.f\_guardarconfig()

    def f\_guardarconfig(self):

        ''' Guardar opciones de configuración de la aplicación '''

        print("Configuración guardada")

    def f\_conectar(self):

        ''' Definir ventana de diálogo para conectar con equipos '''

        print("Conectando")

    def f\_cambiaropc(self):

        ''' Habilitar opción 'Guardar' al elegir alguna opción

            de tipo de conexión, emulador de terminar o

            explorador de archivos '''

        self.menu2.entryconfig("Guardar", state="normal")

    def f\_verestado(self):

        ''' Ocultar o Mostrar barra de estado '''

        if self.estado.get() == 0:

            self.barraest.pack\_forget()

        else:

            self.barraest.pack(side=BOTTOM, fill=X)

    def f\_mostrarmenucontext(self, e):

        ''' Mostrar menú contextual '''

        self.menucontext.post(e.x\_root, e.y\_root)

    def f\_web(self):

        ''' Abrir página web en navegador Internet '''

        pag1 = 'http://python-para-impacientes.blogspot.com/'

        webbrowser.open\_new\_tab(pag1)

    def f\_atajos(self):

        ''' Definir ventana de diálogo con lista de

            combinaciones de teclas de la aplicación '''

        pass

    def f\_acerca(self):

        ''' Definir ventana de diálogo 'Acerca de' '''

        acerca = Toplevel()

        acerca.geometry("320x200")

        acerca.resizable(width=False, height=False)

        acerca.title("Acerca de")

        marco1 = ttk.Frame(acerca, padding=(10, 10, 10, 10),

                           relief=RAISED)

        marco1.pack(side=TOP, fill=BOTH, expand=True)

        etiq1 = Label(marco1, image=self.icono5,

                      relief='raised')

        etiq1.pack(side=TOP, padx=10, pady=10,

                   ipadx=10, ipady=10)

        etiq2 = Label(marco1, text="PyGuidopo "+\_\_version\_\_,

                      foreground='blue', font=self.fuente)

        etiq2.pack(side=TOP, padx=10)

        etiq3 = Label(marco1,

                      text="Python para locos como yo")

        etiq3.pack(side=TOP, padx=10)

        boton1 = Button(marco1, text="Salir",

                        command=acerca.destroy)

        boton1.pack(side=TOP, padx=10, pady=10)

        boton1.focus\_set()

        acerca.transient(self.raiz)

        self.raiz.wait\_window(acerca)

    def f\_salir(self):

        ''' Salir de la aplicación '''

        self.raiz.destroy()

# FUNCIONES DE LA APLICACIÓN

def f\_verificar\_iconos(iconos):

    ''' Verifica existencia de iconos

    iconos -- Lista de iconos '''

    for icono in iconos:

        if not os.path.exists(icono):

            print('Icono no encontrado:', icono)

            return(1)

    return(0)

def main():

    ''' Iniciar aplicación '''

    # INICIALIZAR VARIABLES CON RUTAS

    app\_carpeta = os.getcwd()

    img\_carpeta = app\_carpeta + os.sep + "imagen" + os.sep

    # DECLARAR Y VERIFICAR ICONOS DE LA APLICACIÓN:

    iconos = (img\_carpeta + "pyremoto64x64.png",

              img\_carpeta + "conec16x16.png",

              img\_carpeta + "salir16x16.png",

              img\_carpeta + "star16x16.png",

              img\_carpeta + "conec32x32.png",

              img\_carpeta + "salir32x32.png")

    error1 = f\_verificar\_iconos(iconos)

    if not error1:

        mi\_app = PyRemoto(img\_carpeta, iconos)

    return(0)

if \_\_name\_\_ == '\_\_main\_\_':

    main()

**tener en cuenta guardar las imágenes en una carpeta llamada imagen**

**POR EL TERCER TUTORIAL CREO EL ULTIMO SOBRE ESTE TEMA**

[**https://www.tutorialesprogramacionya.com/pythonya/detalleconcepto.php?punto=58&codigo=58&inicio=45**](https://www.tutorialesprogramacionya.com/pythonya/detalleconcepto.php?punto=58&codigo=58&inicio=45)

tkinter : controles Button y Label

Programa: ejercicio216.py

import tkinter as tk

class Aplicacion:

    def \_\_init\_\_(self):

        self.valor=1

        self.ventana1=tk.Tk()

        self.ventana1.title("Controles Button y Label")

        self.label1=tk.Label(self.ventana1, text=self.valor)

        self.label1.grid(column=0, row=0)

        self.label1.configure(foreground="red")

        self.boton1=tk.Button(self.ventana1, text="Incrementar", command=self.incrementar)

        self.boton1.grid(column=0, row=1)

        self.boton2=tk.Button(self.ventana1, text="Decrementar", command=self.decrementar)

        self.boton2.grid(column=0, row=2)

        self.ventana1.mainloop()

    def incrementar(self):

        self.valor=self.valor+1

        self.label1.config(text=self.valor)

    def decrementar(self):

        self.valor=self.valor-1

        self.label1.config(text=self.valor)

aplicacion1=Aplicacion()

Problema:

Mostrar dos Label, en una se muestra el nombre del programa y en la segunda el año de creación. Disponer un botón para finalizar el programa.  
No permitir al usuario redimensionar la ventana.

Programa: ejercicio217.py

import tkinter as tk

import sys

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.ventana1.title("Prueba")

        self.label1=tk.Label(self.ventana1, text="Sistema de facturación")

        self.label1.grid(column=0, row=0)

        self.label2=tk.Label(self.ventana1, text="2018")

        self.label2.grid(column=0, row=1)

        self.boton1=tk.Button(self.ventana1, text="Finalizar", command=self.finalizar)

        self.boton1.grid(column=0, row=2)

        self.ventana1.resizable(False, False)

        self.ventana1.mainloop()

    def finalizar(self):

        sys.exit(0)

aplicacion1=Aplicacion()

Disponer dos objetos de la clase Button con las etiquetas: "varón" y "mujer", al presionarse mostrar en la barra de títulos de la ventana la etiqueta del botón presionado

EJERCICIO218.PY

import tkinter as tk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.ventana1.title("Prueba")

        self.boton1=tk.Button(self.ventana1, text="Varón", command=self.presionvaron)

        self.boton1.grid(column=0, row=0)

        self.boton2=tk.Button(self.ventana1, text="Mujer", command=self.presionmujer)

        self.boton2.grid(column=1, row=0)

        self.ventana1.mainloop()

    def presionvaron(self):

        self.ventana1.title('Varón')

    def presionmujer(self):

        self.ventana1.title('Mujer')

aplicacion1=Aplicacion()

Mostrar los botones del 1 al 5. Cuando se presiona mostrar en una Label todos los botones presionados hasta ese momento

**Ejercicio219.py**

import tkinter as tk

class Aplicacion:

    def \_\_init\_\_(self):

        self.datos=""

        self.ventana1=tk.Tk()

        self.ventana1.title("Prueba")

        self.boton1=tk.Button(self.ventana1, text="1", command=self.presion1)

        self.boton1.grid(column=0, row=0)

        self.boton2=tk.Button(self.ventana1, text="2", command=self.presion2)

        self.boton2.grid(column=1, row=0)

        self.boton3=tk.Button(self.ventana1, text="3", command=self.presion3)

        self.boton3.grid(column=2, row=0)

        self.boton4=tk.Button(self.ventana1, text="4", command=self.presion4)

        self.boton4.grid(column=3, row=0)

        self.boton5=tk.Button(self.ventana1, text="5", command=self.presion5)

        self.boton5.grid(column=4, row=0)

        self.label1=tk.Label(self.ventana1, text=self.datos)

        self.label1.grid(column=5, row=0)

        self.ventana1.mainloop()

    def presion1(self):

        self.datos=self.datos+"1"

        self.label1.configure(text=self.datos)

    def presion2(self):

        self.datos=self.datos+"2"

        self.label1.configure(text=self.datos)

    def presion3(self):

        self.datos=self.datos+"3"

        self.label1.configure(text=self.datos)

    def presion4(self):

        self.datos=self.datos+"4"

        self.label1.configure(text=self.datos)

    def presion5(self):

        self.datos=self.datos+"5"

        self.label1.configure(text=self.datos)

aplicacion1=Aplicacion()

# tkinter : control Entry

Problema:

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

Programa: ejercicio220.py

import tkinter as tk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.label1=tk.Label(self.ventana1,text="Ingrese un número:")

        self.label1.grid(column=0, row=0)

        self.dato=tk.StringVar()

        self.entry1=tk.Entry(self.ventana1, width=10, textvariable=self.dato)

        self.entry1.grid(column=0, row=1)

        self.boton1=tk.Button(self.ventana1, text="Calcular Cuadrado", command=self.calcularcuadrado)

        self.boton1.grid(column=0, row=2)

        self.label2=tk.Label(self.ventana1,text="resultado")

        self.label2.grid(column=0, row=3)

        self.ventana1.mainloop()

    def calcularcuadrado(self):

        valor=int(self.dato.get())

        cuadrado=valor\*valor

        self.label2.configure(text=cuadrado)

aplicacion1=Aplicacion()

Problema:

Confeccionar un programa que permita ingresar el nombre de usuario en un control Entry y cuando se presione un botón mostrar el valor ingresado en la barra de títulos de la ventana.

Programa: ejercicio221.py

import tkinter as tk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.label1=tk.Label(self.ventana1,text="Ingrese nombre de usuario:")

        self.label1.grid(column=0, row=0)

        self.dato=tk.StringVar()

        self.entry1=tk.Entry(self.ventana1, width=20, textvariable=self.dato)

        self.entry1.grid(column=1, row=0)

        self.boton1=tk.Button(self.ventana1, text="Ingresar", command=self.ingresar)

        self.boton1.grid(column=1, row=1)

        self.ventana1.mainloop()

    def ingresar(self):

        self.ventana1.title(self.dato.get())

aplicacion1=Aplicacion()

Problemas propuestos

* Confeccionar un programa que permita ingresar dos números en controles de tipo Entry, luego sumar los dos valores ingresados y mostrar la suma en una Label al presionar un botón.
* Ingresar el nombre de usuario y clave en controles de tipo Entry. Si se ingresa las cadena (usuario: juan, clave="abc123") luego mostrar en el título de la ventana el mensaje "Correcto" en caso contrario mostrar el mensaje "Incorrecto".  
  Para mostrar '\*' cuando se ingresa la clave debemos pasar en el parámetro 'show' el caracter a mostrar:

**Ejercicio222.py**

import tkinter as tk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.label1=tk.Label(self.ventana1,text="Ingrese primer valor:")

        self.label1.grid(column=0, row=0)

        self.dato1=tk.StringVar()

        self.entry1=tk.Entry(self.ventana1, width=20, textvariable=self.dato1)

        self.entry1.grid(column=1, row=0)

        self.label2=tk.Label(text="Ingrese segundo valor:")

        self.label2.grid(column=0, row=1)

        self.dato2=tk.StringVar()

        self.entry2=tk.Entry(self.ventana1, width=20, textvariable=self.dato2)

        self.entry2.grid(column=1, row=1)

        self.boton1=tk.Button(self.ventana1, text="Sumar", command=self.sumar)

        self.boton1.grid(column=1, row=2)

        self.label3=tk.Label(self.ventana1,text="resultado")

        self.label3.grid(column=1, row=3)

        self.ventana1.mainloop()

    def sumar(self):

        suma=int(self.dato1.get()) + int(self.dato2.get())

        self.label3.configure(text=suma)

aplicacion1=Aplicacion()

**ejercicio223.py**

import tkinter as tk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.label1=tk.Label(self.ventana1,text="Ingrese nombre de usuario:")

        self.label1.grid(column=0, row=0)

        self.dato1=tk.StringVar()

        self.entry1=tk.Entry(self.ventana1, width=30, textvariable=self.dato1)

        self.entry1.grid(column=1, row=0)

        self.label2=tk.Label(text="Ingrese clave:")

        self.label2.grid(column=0, row=1)

        self.dato2=tk.StringVar()

        self.entry2=tk.Entry(self.ventana1, width=30, textvariable=self.dato2, show="\*")

        self.entry2.grid(column=1, row=1)

        self.boton1=tk.Button(self.ventana1, text="Ingresar", command=self.ingresar)

        self.boton1.grid(column=1, row=2)

        self.ventana1.mainloop()

    def ingresar(self):

        if self.dato1.get()=="juan" and self.dato2.get()=="abc123":

            self.ventana1.title("Correcto")

        else:

            self.ventana1.title("Incorrecto")

aplicacion1=Aplicacion()

# tkinter : control Radiobutton

Otro control visual muy común es el Radiobutton que normalmente se muestran un conjunto de Radiobutton y permiten la selección de solo uno de ellos. Se los debe agrupar para que actúen en conjunto, es decir cuando se selecciona uno automáticamente se deben deseleccionar los otros.

Problema:

Mostrar dos controles de tipo Radiobutton con las etiquetas "Varón" y "Mujer", cuando se presione un botón actualizar una Label con el Radiobutton seleccionado.

Programa: ejercicio224.py

import tkinter as tk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.seleccion=tk.IntVar()

        self.seleccion.set(2)

        self.radio1=tk.Radiobutton(self.ventana1,text="Varon", variable=self.seleccion, value=1)

        self.radio1.grid(column=0, row=0)

        self.radio2=tk.Radiobutton(self.ventana1,text="Mujer", variable=self.seleccion, value=2)

        self.radio2.grid(column=0, row=1)

        self.boton1=tk.Button(self.ventana1, text="Mostrar seleccionado", command=self.mostrarseleccionado)

        self.boton1.grid(column=0, row=2)

        self.label1=tk.Label(self.ventana1,text="opcion seleccionada")

        self.label1.grid(column=0, row=3)

        self.ventana1.mainloop()

    def mostrarseleccionado(self):

        if self.seleccion.get()==1:

            self.label1.configure(text="opcion seleccionada=Varon")

        if self.seleccion.get()==2:

            self.label1.configure(text="opcion seleccionada=Mujer")

aplicacion1=Aplicacion()

### Problema:

Disponer dos controles de tipo Entry para el ingreso de enteros. Mediante dos controles Radiobutton permitir seleccionar si queremos sumarlos o restarlos. Al presionar un botón mostrar el resultado de la operación seleccionada.

#### Programa: ejercicio225.py

import tkinter as tk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.label1=tk.Label(self.ventana1,text="Ingrese primer valor:")

        self.label1.grid(column=0, row=0)

        self.dato1=tk.StringVar()

        self.entry1=tk.Entry(self.ventana1, width=20, textvariable=self.dato1)

        self.entry1.grid(column=1, row=0)

        self.label2=tk.Label(self.ventana1,text="Ingrese segundo valor:")

        self.label2.grid(column=0, row=1)

        self.dato2=tk.StringVar()

        self.entry2=tk.Entry(self.ventana1, width=20, textvariable=self.dato2)

        self.entry2.grid(column=1, row=1)

        self.seleccion=tk.IntVar()

        self.radio1=tk.Radiobutton(self.ventana1,text="Sumar", variable=self.seleccion, value=1)

        self.radio1.grid(column=1, row=2)

        self.radio2=tk.Radiobutton(self.ventana1,text="Restar", variable=self.seleccion, value=2)

        self.radio2.grid(column=1, row=3)

        self.boton1=tk.Button(self.ventana1, text="Operar", command=self.operar)

        self.boton1.grid(column=1, row=4)

        self.label3=tk.Label(self.ventana1,text="resultado")

        self.label3.grid(column=1, row=5)

        self.ventana1.mainloop()

    def operar(self):

        if self.seleccion.get()==1:

            suma=int(self.dato1.get())+int(self.dato2.get())

            self.label3.configure(text=suma)

        if self.seleccion.get()==2:

            resta=int(self.dato1.get())-int(self.dato2.get())

            self.label3.configure(text=resta)

aplicacion1=Aplicacion()

Problema propuesto

* Disponer tres controles de tipo Radiobutton con las etiquetas 'Rojo', 'Verde' y 'Azul'. Cuando se presione un botón cambiar el color de fondo del formulario.  
  Si consideramos que la variable ventana1 es un objeto de la clase Tk, luego si queremos que el fondo sea de color rojo debemos llamar al método configure y en el parámetro bg indicar un string con el color a activar ("red", "green" o "blue"):

ejercicio226.py

import tkinter as tk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.seleccion=tk.IntVar()

        self.seleccion.set(1)

        self.radio1=tk.Radiobutton(self.ventana1,text="Rojo", variable=self.seleccion, value=1)

        self.radio1.grid(column=0, row=0)

        self.radio2=tk.Radiobutton(self.ventana1,text="Verde", variable=self.seleccion, value=2)

        self.radio2.grid(column=0, row=1)

        self.radio2=tk.Radiobutton(self.ventana1,text="Azul", variable=self.seleccion, value=3)

        self.radio2.grid(column=0, row=2)

        self.boton1=tk.Button(self.ventana1, text="Cambiar color", command=self.activar)

        self.boton1.grid(column=0, row=3)

        self.ventana1.mainloop()

    def activar(self):

        if self.seleccion.get()==1:

            self.ventana1.configure(bg="red")

        if self.seleccion.get()==2:

            self.ventana1.configure(bg="green")

        if self.seleccion.get()==3:

            self.ventana1.configure(bg="blue")

aplicacion1=Aplicacion()

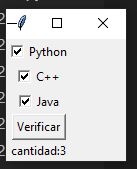
tkinter : control Checkbutton

El control visual Checkbutton permite implementar un botón de dos estados, más conocido como un cuadro de selección.

Problema:

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.

Programa: ejercicio227.py

****

import tkinter as tk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.seleccion1=tk.IntVar()

        self.check1=tk.Checkbutton(self.ventana1,text="Python", variable=self.seleccion1)

        self.check1.grid(column=0, row=0)

        self.seleccion2=tk.IntVar()

        self.check2=tk.Checkbutton(self.ventana1,text="C++", variable=self.seleccion2)

        self.check2.grid(column=0, row=1)

        self.seleccion3=tk.IntVar()

        self.check3=tk.Checkbutton(self.ventana1,text="Java", variable=self.seleccion3)

        self.check3.grid(column=0, row=2)

        self.boton1=tk.Button(self.ventana1, text="Verificar", command=self.verificar)

        self.boton1.grid(column=0, row=4)

        self.label1=tk.Label(self.ventana1,text="cantidad:")

        self.label1.grid(column=0, row=5)

        self.ventana1.mainloop()

    def verificar(self):

        cant=0

        if self.seleccion1.get()==1:

            cant+=1

        if self.seleccion2.get()==1:

            cant+=1

        if self.seleccion3.get()==1:

            cant+=1

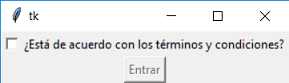
        self.label1.configure(text="cantidad:"+str(cant))

aplicacion1=Aplicacion()

Problema:

Disponer un control Checkbutton que muestre el siguiente mensaje: ¿Está de acuerdo con los términos y condiciones?, además agregar un Button desactivo. Cuando se tilde el Checkbutton inmediatamente activar el botón.

Programa: ejercicio228.py



import tkinter as tk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.seleccion=tk.IntVar()

        self.check1=tk.Checkbutton(self.ventana1,text="¿Está de acuerdo con los términos y condiciones?", variable=self.seleccion, command=self.cambiarestado)

        self.check1.grid(column=0, row=0)

        self.boton1=tk.Button(self.ventana1, text="Entrar", state="disabled", command=self.ingresar)

        self.boton1.grid(column=0, row=1)

        self.ventana1.mainloop()

    def cambiarestado(self):

        if self.seleccion.get()==1:

            self.boton1.configure(state="normal")

        if self.seleccion.get()==0:

            self.boton1.configure(state="disabled")

    def ingresar(self):

        self.ventana1.title("Ingresando...")

aplicacion1=Aplicacion()

Problema propuesto

* Disponer varios objetos de la clase Checkbutton con nombres de navegadores web. En el título de la ventana mostrar todos los nombres de navegadores seleccionados.

ejercicio229.py

****

import tkinter as tk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.seleccion1=tk.IntVar()

        self.check1=tk.Checkbutton(self.ventana1,text="Chrome", variable=self.seleccion1, command=self.cambiartitulo)

        self.check1.grid(column=0, row=0)

        self.seleccion2=tk.IntVar()

        self.check2=tk.Checkbutton(self.ventana1,text="FireFox", variable=self.seleccion2, command=self.cambiartitulo)

        self.check2.grid(column=1, row=0)

        self.seleccion3=tk.IntVar()

        self.check3=tk.Checkbutton(self.ventana1,text="Edge", variable=self.seleccion3, command=self.cambiartitulo)

        self.check3.grid(column=2, row=0)

        self.seleccion4=tk.IntVar()

        self.check4=tk.Checkbutton(self.ventana1,text="Opera", variable=self.seleccion4, command=self.cambiartitulo)

        self.check4.grid(column=3, row=0)

        self.ventana1.mainloop()

    def cambiartitulo(self):

        cadena='';

        if self.seleccion1.get()==1:

            cadena+="Chrome - "

        if self.seleccion2.get()==1:

            cadena+="Firefox - "

        if self.seleccion3.get()==1:

            cadena+="Edge - "

        if self.seleccion4.get()==1:

            cadena+="Opera"

        self.ventana1.title(cadena)

aplicacion1=Aplicacion()

# tkinter : control Listbox

El control visual Listbox se emplea para mostrar una lista de items. El usuario puede seleccionar uno o más elementos de la lista según como se lo configure al crearlo.

### Problema:

Disponer un Listbox con una serie de nombres de frutas. Permitir la selección solo de uno de ellos. Cuando se presione un botón recuperar la fruta seleccionada y mostrarla en una Label.

#### Programa: ejercicio230.py



import tkinter as tk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.listbox1=tk.Listbox(self.ventana1)

        self.listbox1.grid(column=0,row=0)

        self.listbox1.insert(0,"papa")

        self.listbox1.insert(1,"manzana")

        self.listbox1.insert(2,"pera")

        self.listbox1.insert(3,"sandia")

        self.listbox1.insert(4,"naranja")

        self.listbox1.insert(5,"melon")

        self.boton1=tk.Button(self.ventana1, text="Recuperar", command=self.recuperar)

        self.boton1.grid(column=0, row=1)

        self.label1=tk.Label(self.ventana1,text="Seleccionado:")

        self.label1.grid(column=0, row=2)

        self.ventana1.mainloop()

    def recuperar(self):

        if len(self.listbox1.curselection())!=0:

            self.label1.configure(text=self.listbox1.get(self.listbox1.curselection()[0]))

aplicacion1=Aplicacion()

### Problema:

Disponer un Listbox con una serie de nombres de frutas. Permitir la selección de varias frutas. Cuando se presione un botón recuperar todas las frutas seleccionadas y mostrarlas en una Label.

#### Programa: ejercicio231.py



import tkinter as tk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.listbox1=tk.Listbox(self.ventana1, selectmode=tk.MULTIPLE)

        self.listbox1.grid(column=0,row=0)

        self.listbox1.insert(0,"papa")

        self.listbox1.insert(1,"manzana")

        self.listbox1.insert(2,"pera")

        self.listbox1.insert(3,"sandia")

        self.listbox1.insert(4,"naranja")

        self.listbox1.insert(5,"melon")

        self.boton1=tk.Button(self.ventana1, text="Recuperar", command=self.recuperar)

        self.boton1.grid(column=0, row=1)

        self.label1=tk.Label(self.ventana1,text="Seleccionado:")

        self.label1.grid(column=0, row=2)

        self.ventana1.mainloop()

    def recuperar(self):

        if len(self.listbox1.curselection())!=0:

            todas=''

            for posicion in self.listbox1.curselection():

                todas+=self.listbox1.get(posicion)+"\n"

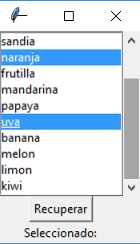
            self.label1.configure(text=todas)

aplicacion1=Aplicacion()

### Barra de scroll

Por defecto no aparece una barra de scroll si la cantidad de item supera el tamaño del cuadro del Listbox. Para que se muestre una barra de scroll la debemos crear y enlazar con el Listbox.  
El mismo programa anterior pero con la barra de scroll queda:

#### Programa: ejercicio232.py



import tkinter as tk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.scroll1 = tk.Scrollbar(self.ventana1, orient=tk.VERTICAL)

        self.listbox1=tk.Listbox(self.ventana1, selectmode=tk.MULTIPLE, yscrollcommand=self.scroll1.set)

        self.listbox1.grid(column=0,row=0)

        self.scroll1.configure(command=self.listbox1.yview)

        self.scroll1.grid(column=1, row=0, sticky='NS')

        self.listbox1.insert(0,"papa")

        self.listbox1.insert(1,"manzana")

        self.listbox1.insert(2,"pera")

        self.listbox1.insert(3,"sandia")

        self.listbox1.insert(4,"naranja")

        self.listbox1.insert(5,"melon")

        self.listbox1.insert(6,"limon")

        self.listbox1.insert(7,"kiwi")

        self.listbox1.insert(5,"banana")

        self.listbox1.insert(5,"uva")

        self.listbox1.insert(5,"papaya")

        self.listbox1.insert(5,"mandarina")

        self.listbox1.insert(5,"frutilla")

        self.boton1=tk.Button(self.ventana1, text="Recuperar", command=self.recuperar)

        self.boton1.grid(column=0, row=1)

        self.label1=tk.Label(self.ventana1,text="Seleccionado:")

        self.label1.grid(column=0, row=2)

        self.ventana1.mainloop()

    def recuperar(self):

        if len(self.listbox1.curselection())!=0:

            todas=''

            for posicion in self.listbox1.curselection():

                todas+=self.listbox1.get(posicion)+"\n"

            self.label1.configure(text=todas)

aplicacion1=Aplicacion()

Problema propuesto

* Solicitar el ingreso del nombre de una persona y seleccionar de un control Listbox un país. Al presionar un botón mostrar en la barra de la ventana el nombre ingresado y el país seleccionado

[**https://www.youtube.com/watch?v=oTCkZBGjU3s**](https://www.youtube.com/watch?v=oTCkZBGjU3s)

ejercicio233.py

mport tkinter as tk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.label1=tk.Label(self.ventana1,text="Ingrese nombre")

        self.label1.grid(column=0, row=0)

        self.nombre=tk.StringVar()

        self.entry1=tk.Entry(self.ventana1, width=40, textvariable=self.nombre)

        self.entry1.grid(column=0, row=1)

        self.label2=tk.Label(self.ventana1,text="Seleccione país")

        self.label2.grid(column=0, row=2)

        self.listbox1=tk.Listbox(self.ventana1)

        self.listbox1.grid(column=0,row=3)

        self.listbox1.insert(0,"Argentina")

        self.listbox1.insert(1,"Chile")

        self.listbox1.insert(2,"Bolivia")

        self.listbox1.insert(3,"Paraguay")

        self.listbox1.insert(4,"Brasil")

        self.listbox1.insert(5,"Uruguay")

        self.boton1=tk.Button(self.ventana1, text="Recuperar", command=self.mostrardatos)

        self.boton1.grid(column=0, row=4)

        self.ventana1.mainloop()

    def mostrardatos(self):

         if len(self.listbox1.curselection())!=0:

            self.ventana1.title("Nombre:"+self.nombre.get()+" Pais:"+self.listbox1.get(self.listbox1.curselection()[0]))

aplicacion1=Aplicacion()

módulo ttk

Tk hemos dicho que es una biblioteca de controles visuales que los podemos acceder desde Python y desde otros lenguajes de programación.

En la versión Tk 8.5 sumó una nueva serie de controles visuales ( Notebook, Combobox etc.) y modernizó los que hemos visto en los conceptos anteriores. Para hacer uso de esta nueva versión de la biblioteca en Python se implementó un nuevo módulo y se lo agregó al paquete tkinter.

Para hacer uso de este conjunto de Widget (controles visuales) debemos importar el paquete ttk.

Todo lo que conocemos hasta ahora de los controles visuales del módulo tkinter funciona prácticamente sin cambios, lo que deberemos hacer es crear objetos de la clase Button, Entry etc. recuperándolos ahora del módulo tkinter.ttk

### Importar el módulo ttk

Debemos utilizar la siguiente sintaxis para importar el nuevo módulo:

from tkinter import ttk

### Problema:

Mostrar una ventana y en su interior dos botones y una label utilizando el módulo ttk. La label muestra inicialmente el valor 1. Cada uno de los botones permiten incrementar o decrementar en uno el contenido de la label

Si bien los cambios visuales que aparecen en la clase Button no son significativos el módulo tkinter.ttk trae nuevas funcionalidades, nuevos controles y permite que nuestra interfaz visual se adapte al sistema operativo donde se está ejecutando (Windows, Mac, Linux etc.)

Analicemos que cambios debemos disponer a nuestra aplicación para utilizar las componentes Button y Label del módulo tkinter.ttk.

Debemos seguir importando el paquete tkinter como lo veníamos haciendo:

#### Programa: ejercicio234.py

import tkinter as tk

from tkinter import ttk

class Aplicacion:

    def \_\_init\_\_(self):

        self.valor=1

        self.ventana1=tk.Tk()

        self.ventana1.title("Controles Button y Label")

        self.label1=ttk.Label(self.ventana1, text=self.valor)

        self.label1.grid(column=0, row=0)

        self.label1.configure(foreground="red")

        self.boton1=ttk.Button(self.ventana1, text="Incrementar", command=self.incrementar)

        self.boton1.grid(column=0, row=1)

        self.boton2=ttk.Button(self.ventana1, text="Decrementar", command=self.decrementar)

        self.boton2.grid(column=0, row=2)

        self.ventana1.mainloop()

    def incrementar(self):

        self.valor=self.valor+1

        self.label1.config(text=self.valor)

    def decrementar(self):

        self.valor=self.valor-1

        self.label1.config(text=self.valor)

aplicacion1=Aplicacion()

Resolvamos otros problemas ya resueltos empleando el nuevo módulo de Widged propuesto en el módulo tkinter.ttk.

### Problema:

Ingresar el nombre de usuario y clave en controles de tipo Entry. Si se ingresa las cadena (usuario: juan, clave="abc123") luego mostrar en el título de la ventana el mensaje "Correcto" en caso contrario mostrar el mensaje "Incorrecto". Utilizar Widget del módulo ttk.

#### Programa: ejercicio235.py

import tkinter as tk

from tkinter import ttk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.label1=ttk.Label(text="Ingrese nombre de usuario:")

        self.label1.grid(column=0, row=0)

        self.dato1=tk.StringVar()

        self.entry1=ttk.Entry(self.ventana1, width=30, textvariable=self.dato1)

        self.entry1.grid(column=1, row=0)

        self.label2=ttk.Label(text="Ingrese clave:")

        self.label2.grid(column=0, row=1)

        self.dato2=tk.StringVar()

        self.entry2=ttk.Entry(self.ventana1, width=30, textvariable=self.dato2, show="\*")

        self.entry2.grid(column=1, row=1)

        self.boton1=ttk.Button(self.ventana1, text="Ingresar", command=self.ingresar)

        self.boton1.grid(column=1, row=2)

        self.ventana1.mainloop()

    def ingresar(self):

        if self.dato1.get()=="juan" and self.dato2.get()=="abc123":

            self.ventana1.title("Correcto")

        else:

            self.ventana1.title("Incorrecto")

aplicacion1=Aplicacion()

### Problema:

Mostrar dos controles de tipo Radiobutton con las etiquetas "Varón" y "Mujer", cuando se presione un botón actualizar una Label con el Radiobutton seleccionado.

#### Programa: ejercicio236.py

import tkinter as tk

from tkinter import ttk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.seleccion=tk.IntVar()

        self.seleccion.set(2)

        self.radio1=ttk.Radiobutton(self.ventana1,text="Varon", variable=self.seleccion, value=1)

        self.radio1.grid(column=0, row=0)

        self.radio2=ttk.Radiobutton(self.ventana1,text="Mujer", variable=self.seleccion, value=2)

        self.radio2.grid(column=0, row=1)

        self.boton1=ttk.Button(self.ventana1, text="Mostrar seleccionado", command=self.mostrarseleccionado)

        self.boton1.grid(column=0, row=2)

        self.label1=ttk.Label(text="opcion seleccionada")

        self.label1.grid(column=0, row=3)

        self.ventana1.mainloop()

    def mostrarseleccionado(self):

        if self.seleccion.get()==1:

            self.label1.configure(text="opcion seleccionada=Varon")

        if self.seleccion.get()==2:

            self.label1.configure(text="opcion seleccionada=Mujer")

aplicacion1=Aplicacion()

### Problema:

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. Utilizar Widget del módulo ttk.

#### Programa: ejercicio237.py

import tkinter as tk

from tkinter import ttk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.seleccion1=tk.IntVar()

        self.check1=ttk.Checkbutton(self.ventana1,text="Python", variable=self.seleccion1)

        self.check1.grid(column=0, row=0)

        self.seleccion2=tk.IntVar()

        self.check2=ttk.Checkbutton(self.ventana1,text="C++", variable=self.seleccion2)

        self.check2.grid(column=0, row=1)

        self.seleccion3=tk.IntVar()

        self.check3=ttk.Checkbutton(self.ventana1,text="Java", variable=self.seleccion3)

        self.check3.grid(column=0, row=2)

        self.boton1=ttk.Button(self.ventana1, text="Verificar", command=self.verificar)

        self.boton1.grid(column=0, row=4)

        self.label1=ttk.Label(text="cantidad:")

        self.label1.grid(column=0, row=5)

        self.ventana1.mainloop()

    def verificar(self):

        cant=0

        if self.seleccion1.get()==1:

            cant+=1

        if self.seleccion2.get()==1:

            cant+=1

        if self.seleccion3.get()==1:

            cant+=1

        self.label1.configure(text="cantidad:"+str(cant))

aplicacion1=Aplicacion()

### Problema:

Disponer un Listbox con una serie de nombres de frutas. Permitir la selección solo de uno de ellos. Cuando se presione un botón recuperar la fruta seleccionada y mostrarla en una Label.

#### Programa: ejercicio238.py

import tkinter as tk

from tkinter import ttk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.listbox1=tk.Listbox(self.ventana1)

        self.listbox1.grid(column=0,row=0)

        self.listbox1.insert(0,"papa")

        self.listbox1.insert(1,"manzana")

        self.listbox1.insert(2,"pera")

        self.listbox1.insert(3,"sandia")

        self.listbox1.insert(4,"naranja")

        self.listbox1.insert(5,"melon")

        self.boton1=ttk.Button(self.ventana1, text="Recuperar", command=self.recuperar)

        self.boton1.grid(column=0, row=1)

        self.label1=ttk.Label(text="Seleccionado:")

        self.label1.grid(column=0, row=2)

        self.ventana1.mainloop()

    def recuperar(self):

        if len(self.listbox1.curselection())!=0:

            self.label1.configure(text=self.listbox1.get(self.listbox1.curselection()[0]))

aplicacion1=Aplicacion()

# ttk : control Combobox

El control Combobox del paquete ttk permite seleccionar un string de un conjunto de items que se despliegan.

Podemos indicar cual elemento muestre por defecto mediante el método 'current'.

Por defecto el operador puede además de seleccionar un elemento cargar por teclado cualquier cadena.

Problema:

Mostrar en una ventana un control de tipo Combobox con los días de la semana. Cuando se presione un botón actualizar una Label con el día seleccionado.

Programa: ejercicio239.py

import tkinter as tk

from tkinter import ttk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.label1=ttk.Label(self.ventana1, text="Seleccione un día de la semana")

        self.label1.grid(column=0, row=0)

        self.opcion=tk.StringVar()

        diassemana=("lunes","martes","miércoles","jueves","viernes","sábado","domingo")

        self.combobox1=ttk.Combobox(self.ventana1,

                                  width=10,

                                  textvariable=self.opcion,

                                  values=diassemana)

        self.combobox1.current(0)

        self.combobox1.grid(column=0, row=1)

        self.boton1=tk.Button(self.ventana1, text="Recuperar", command=self.recuperar)

        self.boton1.grid(column=0, row=2)

        self.label2=ttk.Label(self.ventana1, text="Día seleccionado:")

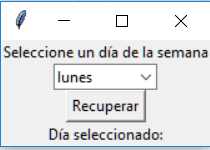
        self.label2.grid(column=0, row=3)

        self.ventana1.mainloop()

    def recuperar(self):

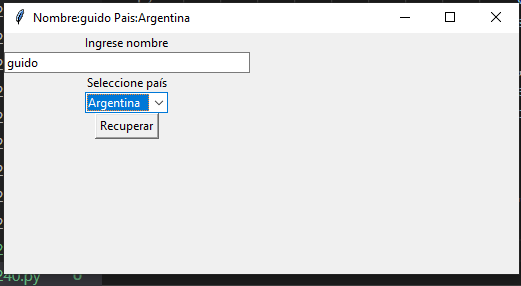
        self.label2.configure(text=self.opcion.get())

aplicacion1=Aplicacion()



Problema propuesto

* Solicitar el ingreso del nombre de una persona y seleccionar de un control Combobox un país. Al presionar un botón mostrar en la barra de la ventana el nombre ingresado y el país seleccionado.
* ejercicio240.py
* import tkinter as tk
* from tkinter import ttk
* class Aplicacion:
* def \_\_init\_\_(self):
* self.ventana1=tk.Tk()
* self.label1=ttk.Label(text="Ingrese nombre")
* self.label1.grid(column=0, row=0)
* self.nombre=tk.StringVar()
* self.entry1=ttk.Entry(self.ventana1, width=40, textvariable=self.nombre)
* self.entry1.grid(column=0, row=1)
* self.label2=ttk.Label(text="Seleccione país")
* self.label2.grid(column=0, row=2)
* self.pais=tk.StringVar()
* paises=("Argentina","Chile","Bolivia","Paraguay","Brasil","Uruguay")
* self.combobox1=ttk.Combobox(self.ventana1,
* width=10,
* textvariable=self.pais,
* values=paises,
* state='readonly')
* #self.combobox1.current(0)
* self.combobox1.grid(column=0, row=3)
* self.boton1=tk.Button(self.ventana1, text="Recuperar", command=self.mostrardatos)
* self.boton1.grid(column=0, row=4)
* self.ventana1.mainloop()
* def mostrardatos(self):
* self.ventana1.title("Nombre:"+self.nombre.get()+" Pais:"+self.combobox1.get())
* aplicacion1=Aplicacion()

****

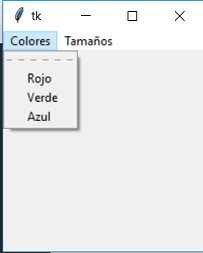
tkinter : control Menu

Para implementar los típicos menú de barra horizontales que aparecen en las aplicaciones cuando utilizamos la librería Tk necesitamos crear objetos de la clase Menu que se encuentra declarada en el paquete tkinter y no en el paquete tkinter.ttk.

Veremos con un ejemplo implementar un menú con una serie de opciones.

### Problema:

Confeccionar una aplicación que muestre dos opciones en el menú de barra superior. La primer opción despliega un submenú que permita cambiar el color de fondo del formulario y la segunda permita cambiar el tamaño de formulario:



#### Programa: ejercicio241.py

import tkinter as tk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        menubar1 = tk.Menu(self.ventana1)

        self.ventana1.config(menu=menubar1)

        opciones1 = tk.Menu(menubar1)

        opciones1.add\_command(label="Rojo", command=self.fijarrojo)

        opciones1.add\_command(label="Verde", command=self.fijarverde)

        opciones1.add\_command(label="Azul", command=self.fijarazul)

        menubar1.add\_cascade(label="Colores", menu=opciones1)

        opciones2 = tk.Menu(menubar1)

        opciones2.add\_command(label="640x480", command=self.ventanachica)

        opciones2.add\_command(label="1024x800", command=self.ventanagrande)

        menubar1.add\_cascade(label="Tamaños", menu=opciones2)

        self.ventana1.mainloop()

    def fijarrojo(self):

        self.ventana1.configure(background="red")

    def fijarverde(self):

        self.ventana1.configure(background="green")

    def fijarazul(self):

        self.ventana1.configure(background="blue")

    def ventanachica(self):

        self.ventana1.geometry("640x480")

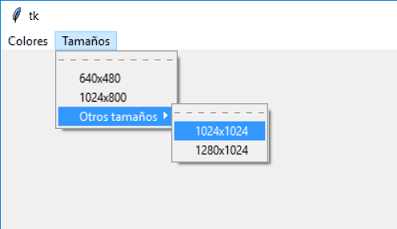
    def ventanagrande(self):

        self.ventana1.geometry("1024x800")

aplicacion1=Aplicacion()

Variantes en un menú.

* Podemos agregar líneas separadoras entre opciones mediante la llamada al método 'add\_separator':
* Los menú desplegables muestran una línea de puntos antes de la primer opción. Si el operador hace clic sobre la misma el menú desplegable se muestra en una barra de herramientas fuera de la ventana principal (puede tener sentido si en un programa debemos seleccionar opciones en forma muy seguida y no queremos tener que estar abriendo el menú desplegable):
* Si queremos evitar esta característica de los menú desplegables de tkinter debemos pasar el parámetro 'tearoff' con el valor 0:
* opciones1 = tk.Menu(menubar1, tearoff=0)
* opciones1.add\_command(label="Rojo", command=self.fijarrojo)
* opciones1.add\_command(label="Verde", command=self.fijarverde)
* opciones1.add\_separator()
* opciones1.add\_command(label="Azul", command=self.fijarazul)
* Podemos comprobar que no aparece la línea de puntos:
* Otra posibilidad es agregar teclas de acceso rápido a opciones de nuestro menú. Esto se resuelve la parte visual agregando el parámetro 'acelerator', y por otro lado asignar a cada una de las teclas de acceso rápido la ejecución de un método:



#### ejercicio241b.py

import tkinter as tk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        menubar1 = tk.Menu(self.ventana1)

        self.ventana1.config(menu=menubar1)

        opciones1 = tk.Menu(menubar1, tearoff=0)

        opciones1.add\_command(label="Rojo", command=self.fijarrojo, accelerator="Ctrl+R")

        opciones1.add\_command(label="Verde", command=self.fijarverde, accelerator="Ctrl+V")

        opciones1.add\_separator()

        opciones1.add\_command(label="Azul", command=self.fijarazul, accelerator="Ctrl+A")

        self.ventana1.bind\_all("<Control-r>", self.cambiar)

        self.ventana1.bind\_all("<Control-v>", self.cambiar)

        self.ventana1.bind\_all("<Control-a>", self.cambiar)

        menubar1.add\_cascade(label="Colores", menu=opciones1)

        opciones2 = tk.Menu(menubar1)

        opciones2.add\_command(label="640x480", command=self.ventanachica)

        opciones2.add\_command(label="1024x800", command=self.ventanagrande)

        menubar1.add\_cascade(label="Tamaños", menu=opciones2)

        self.ventana1.mainloop()

    def cambiar(self, event):

        if event.keysym=="r":

            self.fijarrojo()

        if event.keysym=="v":

            self.fijarverde()

        if event.keysym=="a":

            self.fijarazul()

    def fijarrojo(self):

        self.ventana1.configure(background="red")

    def fijarverde(self):

        self.ventana1.configure(background="green")

    def fijarazul(self):

        self.ventana1.configure(background="blue")

    def ventanachica(self):

        self.ventana1.geometry("640x480")

    def ventanagrande(self):

        self.ventana1.geometry("1024x800")

aplicacion1=Aplicacion()

#### ejercicio241c.py

import tkinter as tk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        menubar1 = tk.Menu(self.ventana1)

        self.ventana1.config(menu=menubar1)

        opciones1 = tk.Menu(menubar1)

        opciones1.add\_command(label="Rojo", command=self.fijarrojo)

        opciones1.add\_command(label="Verde", command=self.fijarverde)

        opciones1.add\_command(label="Azul", command=self.fijarazul)

        menubar1.add\_cascade(label="Colores", menu=opciones1)

        opciones2 = tk.Menu(menubar1)

        opciones2.add\_command(label="640x480", command=self.ventanachica)

        opciones2.add\_command(label="1024x800", command=self.ventanagrande)

        submenu1=tk.Menu(menubar1)

        submenu1.add\_command(label="1024x1024", command=self.tamano1)

        submenu1.add\_command(label="1280x1024", command=self.tamano2)

        opciones2.add\_cascade(label="Otros tamaños", menu= submenu1)

        menubar1.add\_cascade(label="Tamaños", menu=opciones2)

        self.ventana1.mainloop()

    def fijarrojo(self):

        self.ventana1.configure(background="red")

    def fijarverde(self):

        self.ventana1.configure(background="green")

    def fijarazul(self):

        self.ventana1.configure(background="blue")

    def ventanachica(self):

        self.ventana1.geometry("640x480")

    def ventanagrande(self):

        self.ventana1.geometry("1024x800")

    def tamano1(self):

        self.ventana1.geometry("1024x1024")

    def tamano2(self):

        self.ventana1.geometry("1280x1024")

aplicacion1=Aplicacion()

Problema propuesto

* Mediante dos controles de tipo Entry permitir el ingreso de dos números. Crear un menú que contenga una opción que cambie el tamaño de la ventana con los valores ingresados por teclado. Finalmente disponer otra opción que finalice el programa

ejercicio242.py

import tkinter as tk

from tkinter import ttk

import sys

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        menubar1 = tk.Menu(self.ventana1)

        self.ventana1.config(menu=menubar1)

        opciones1 = tk.Menu(menubar1)

        opciones1.add\_command(label="Cambiar dimensión ventana", command=self.fijartamano)

        opciones1.add\_command(label="Finalizar", command=self.finalizar)

        menubar1.add\_cascade(label="Opciones", menu=opciones1)

        self.label1=ttk.Label(text="Ingrese ancho de la ventana en X:")

        self.label1.grid(column=0, row=0)

        self.ancho=tk.StringVar()

        self.entry1=ttk.Entry(self.ventana1, width=10, textvariable=self.ancho)

        self.entry1.grid(column=0, row=1)

        self.label2=ttk.Label(text="Ingrese ancho de la ventana en Y:")

        self.label2.grid(column=0, row=2)

        self.alto=tk.StringVar()

        self.entry2=ttk.Entry(self.ventana1, width=10, textvariable=self.alto)

        self.entry2.grid(column=0, row=3)

        self.ventana1.mainloop()

    def fijartamano(self):

        self.ventana1.geometry(self.ancho.get()+"x"+self.alto.get())

    def finalizar(self):

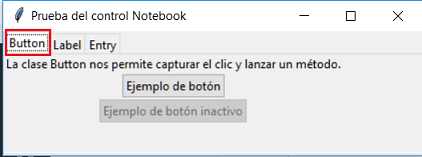
        sys.exit()

aplicacion1=Aplicacion()

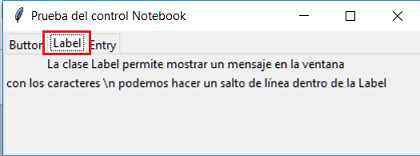
ttk : controles Notebook y Frame

La clase Notebook nos permite crear un cuaderno con una serie de pestañas en la parte superior. En cada pestaña asociamos un objeto de la clase Frame y dentro de esta podemos disponer distintos controles visuales que hemos visto hasta ahora como pueden ser Label, Button, Radiobutton, Checkbutton, Entry etc.

Un ejemplo visual de un Notebook es:



Según la pestaña seleccionada se muestra un Frame con distintos controles visuales:



Problema:

Confeccionar una aplicación que muestre un cuaderno con tres pestañas. Los títulos de cada pestaña deben ser 'Button', 'Label' y 'Entry'. Según la pestaña seleccionada mostrar un mensaje informando el objetivo de la clase y un ejemplo de la misma.

Programa: ejercicio243.py

import tkinter as tk

from tkinter import ttk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.ventana1.title("Prueba del control Notebook")

        self.cuaderno1 = ttk.Notebook(self.ventana1)

        self.pagina1 = ttk.Frame(self.cuaderno1)

        self.cuaderno1.add(self.pagina1, text="Button")

        self.label1=ttk.Label(self.pagina1, text="La clase Button nos permite capturar el clic y lanzar un método.")

        self.label1.grid(column=0, row=0)

        self.boton1=ttk.Button(self.pagina1, text="Ejemplo de botón")

        self.boton1.grid(column=0, row=1)

        self.boton2=ttk.Button(self.pagina1, text="Ejemplo de botón inactivo", state="disabled")

        self.boton2.grid(column=0, row=2)

        self.pagina2 = ttk.Frame(self.cuaderno1)

        self.cuaderno1.add(self.pagina2, text="Label")

        self.label2=ttk.Label(self.pagina2, text="La clase Label permite mostrar un mensaje en la ventana")

        self.label2.grid(column=0, row=0)

        self.label3=ttk.Label(self.pagina2, text="con los caracteres \\n podemos hacer un salto de línea dentro de la Label")

        self.label3.grid(column=0, row=1)

        self.pagina3 = ttk.Frame(self.cuaderno1)

        self.cuaderno1.add(self.pagina3, text="Entry")

        self.label4=ttk.Label(self.pagina3, text="""En tkinter el control de entrada de datos por teclado se llama Entry.\n

Con este control aparece el típico recuadro que cuando se le da foco aparece el cursor en forma intermitente\n

esperando que el operador escriba algo por teclado.""")

        self.label4.grid(column=0, row=0)

        self.entry1=tk.Entry(self.pagina3, width=30)

        self.entry1.grid(column=0, row=1)

        self.cuaderno1.grid(column=0, row=0)

        self.ventana1.mainloop()

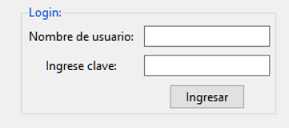
aplicacion1=Aplicacion()

ttk : control LabelFrame

El control visual LabelFrame tiene una funcionalidad casi idéntica a la componente Frame que vimos en el concepto anterior, la única diferencia es que agrega un texto en la parte superior del Frame y hace un recuadro alrededor del mismo.

El Widget LabelFrame es un contenedor donde podemos agregar en su interior otros Widget como Button, Label, Entry, Radiobutton etc.

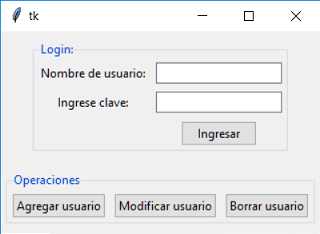
La representación visual de un LabelFrame que tiene 2 Label, 2 Entry y un Button es:



Problema:

Confeccionar una aplicación que muestre dos controles de tipo LabelFrame. En la primera disponer 2 Label, 2 Entry y un Button, en el segundo LabelFrame disponer 3 botones.

La representación visual debe ser:



Programa: ejercicio244.py

import tkinter as tk

from tkinter import ttk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.labelframe1=ttk.LabelFrame(self.ventana1, text="Login:")

        self.labelframe1.grid(column=0, row=0, padx=5, pady=10)

        self.login()

        self.labelframe2=ttk.LabelFrame(self.ventana1, text="Operaciones")

        self.labelframe2.grid(column=0, row=1, padx=5, pady=10)

        self.operaciones()

        self.ventana1.mainloop()

    def login(self):

        self.label1=ttk.Label(self.labelframe1, text="Nombre de usuario:")

        self.label1.grid(column=0, row=0, padx=4, pady=4)

        self.entry1=ttk.Entry(self.labelframe1)

        self.entry1.grid(column=1, row=0, padx=4, pady=4)

        self.label2=ttk.Label(self.labelframe1, text="Ingrese clave:")

        self.label2.grid(column=0, row=1, padx=4, pady=4)

        self.entry2=ttk.Entry(self.labelframe1, show="\*")

        self.entry2.grid(column=1, row=1, padx=4, pady=4)

        self.boton1=ttk.Button(self.labelframe1, text="Ingresar")

        self.boton1.grid(column=1, row=2, padx=4, pady=4)

    def operaciones(self):

        self.boton2=ttk.Button(self.labelframe2, text="Agregar usuario")

        self.boton2.grid(column=0, row=0, padx=4, pady=4)

        self.boton3=ttk.Button(self.labelframe2, text="Modificar usuario")

        self.boton3.grid(column=1, row=0, padx=4, pady=4)

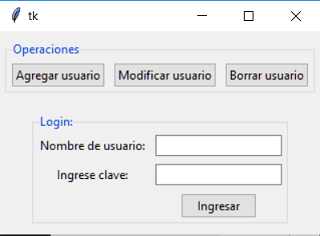
        self.boton4=ttk.Button(self.labelframe2, text="Borrar usuario")

        self.boton4.grid(column=2, row=0, padx=4, pady=4)

aplicacion1=Aplicacion()

El empleo de los LabelFrame nos permite crear interfaces visuales más claras para los operadores de nuestro programa.

Si tenemos luego que hacer cambios de ubicación de los LabelFrame es muy sencillo, solo cambiamos los valores que le pasamos a grid. Si queremos que primero se muestre la barra de botones y luego el login:



Esta nueva representación visual se obtiene solo cambiando dos números de fila:

def \_\_init\_\_(self):

self.ventana1=tk.Tk()

self.labelframe1=ttk.LabelFrame(self.ventana1, text="Login:")

self.labelframe1.grid(column=0, row=**1**, padx=5, pady=10)

self.login()

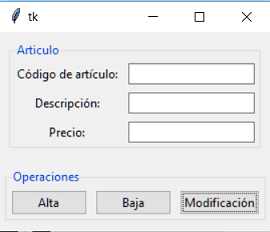
self.labelframe2=ttk.LabelFrame(self.ventana1, text="Operaciones")

self.labelframe2.grid(column=0, row=**0**, padx=5, pady=10)

self.operaciones()

self.ventana1.mainloop()

Problema propuesto

* Mediante dos controles de tipo LabelFrame implementar la siguiente interfaz visual:  
  

ejercicio245.py

import tkinter as tk

from tkinter import ttk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.labelframe1=ttk.LabelFrame(self.ventana1, text="Articulo")

        self.labelframe1.grid(column=0, row=0, padx=5, pady=10)

        self.form\_articulo()

        self.labelframe2=ttk.LabelFrame(self.ventana1, text="Operaciones")

        self.labelframe2.grid(column=0, row=1, padx=5, pady=10)

        self.form\_operaciones()

        self.ventana1.mainloop()

    def form\_articulo(self):

        self.label1=ttk.Label(self.labelframe1, text="Código de artículo:")

        self.label1.grid(column=0, row=0, padx=4, pady=4)

        self.entry1=ttk.Entry(self.labelframe1)

        self.entry1.grid(column=1, row=0, padx=4, pady=4)

        self.label2=ttk.Label(self.labelframe1, text="Descripción:")

        self.label2.grid(column=0, row=1, padx=4, pady=4)

        self.entry2=ttk.Entry(self.labelframe1, show="\*")

        self.entry2.grid(column=1, row=1, padx=4, pady=4)

        self.label3=ttk.Label(self.labelframe1, text="Precio:")

        self.label3.grid(column=0, row=2, padx=4, pady=4)

        self.entry3=ttk.Entry(self.labelframe1, show="\*")

        self.entry3.grid(column=1, row=2, padx=4, pady=4)

    def form\_operaciones(self):

        self.boton1=ttk.Button(self.labelframe2, text="Alta")

        self.boton1.grid(column=0, row=0, padx=4, pady=4)

        self.boton2=ttk.Button(self.labelframe2, text="Baja")

        self.boton2.grid(column=1, row=0, padx=4, pady=4)

        self.boton3=ttk.Button(self.labelframe2, text="Modificación")

        self.boton3.grid(column=2, row=0, padx=4, pady=4)

aplicacion1=Aplicacion()

# tkinter : Layout Manager (administrador de diseño)

Una de las herramientas fundamentales cuando armamos interfaces visuales es la metodología que utilizamos para disponer los controles dentro del formulario. Hasta ahora hemos utilizado el administrador de diseño Grid.

En la librería GUI tkinter disponemos de tres Layout Manager para disponer controles dentro de una ventana:

1. Grid
2. Pack
3. Place

Solo se puede utilizar uno de estos Layout Manager dentro de un contenedor, recordemos que un contenedor puede ser la ventana propiamente dicha, un Frame o un LabelFrame.

El gestor de diseño más completo y que se adapta en la mayoría de las situaciones es el Grid, pero podemos en muchos casos crear Frame o LabelFrame y definir dentro de estos Layout Manager de tipo Pack o Place.

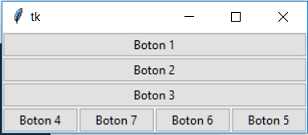
### Layout Manager: Pack

Veamos con un ejemplo como se ubican los Widget utilizando Pack.

### Problema:

Disponer una serie de botones utilizando el Layout Manager de tipo Pack.

La representación visual debe ser:



#### Programa: ejercicio246.py

import tkinter as tk

from tkinter import ttk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.boton1=ttk.Button(self.ventana1, text="Boton 1")

        self.boton1.pack(side=tk.TOP, fill=tk.BOTH)

        self.boton2=ttk.Button(self.ventana1, text="Boton 2")

        self.boton2.pack(side=tk.TOP, fill=tk.BOTH)

        self.boton3=ttk.Button(self.ventana1, text="Boton 3")

        self.boton3.pack(side=tk.TOP, fill=tk.BOTH)

        self.boton4=ttk.Button(self.ventana1, text="Boton 4")

        self.boton4.pack(side=tk.LEFT)

        self.boton5=ttk.Button(self.ventana1, text="Boton 5")

        self.boton5.pack(side=tk.RIGHT)

        self.boton6=ttk.Button(self.ventana1, text="Boton 6")

        self.boton6.pack(side=tk.RIGHT)

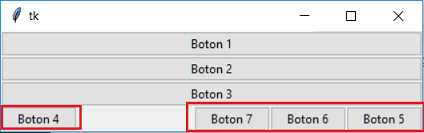
        self.boton7=ttk.Button(self.ventana1, text="Boton 7")

        self.boton7.pack(side=tk.RIGHT)

        self.ventana1.mainloop()

aplicacion1=Aplicacion()

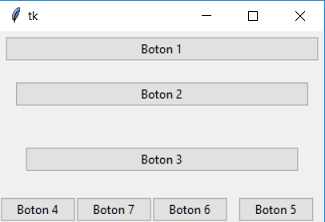
Si agrandamos la ventana podemos ver que el botón fijado a izquierda siempre permanece en dicho lugar y los fijados a derecha se desplazan:



El parámetro side puede recibir alguno de estos cuatro valores:

* tk.TOP
* tk.LEFT
* tk.RIGHT
* tk.BOTTOM

Podemos pasar los parámetros padx y pady para dejar espacio entre los controles visuales y el borde del contenedor:



Para lograr ese resultado debemos hacer los siguientes cambios:

import tkinter as tk

from tkinter import ttk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.boton1=ttk.Button(self.ventana1, text="Boton 1")

        self.boton1.pack(side=tk.TOP, fill=tk.BOTH, padx=5, pady=5)

        self.boton2=ttk.Button(self.ventana1, text="Boton 2")

        self.boton2.pack(side=tk.TOP, fill=tk.BOTH, padx=15, pady=15)

        self.boton3=ttk.Button(self.ventana1, text="Boton 3")

        self.boton3.pack(side=tk.TOP, fill=tk.BOTH, padx=25, pady=25)

        self.boton4=ttk.Button(self.ventana1, text="Boton 4")

        self.boton4.pack(side=tk.LEFT)

        self.boton5=ttk.Button(self.ventana1, text="Boton 5")

        self.boton5.pack(side=tk.RIGHT, padx=10)

        self.boton6=ttk.Button(self.ventana1, text="Boton 6")

        self.boton6.pack(side=tk.RIGHT)

        self.boton7=ttk.Button(self.ventana1, text="Boton 7")

        self.boton7.pack(side=tk.RIGHT)

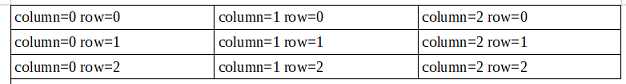
        self.ventana1.mainloop()

aplicacion1=Aplicacion()

### Layout Manager: Grid

Este tipo de Layout Manager lo hemos estado utilizando en muchos conceptos anteriores, veremos otras posibilidades que nos suministra.

Este tipo de Layout define una tabla con columnas y filas, cada vez que agregamos un Widget indicamos en que columna y fila se debe ubicar:

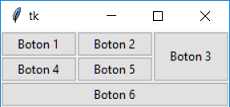


Un tema que no vimos es que podemos expandir celdas de la tabla para que se ocupen más de una columna o más de una fila.

### Problema:

Disponer una serie de botones utilizando el Layout Manager de tipo Grid.

La representación visual debe ser:



#### Programa: ejercicio247.py

import tkinter as tk

from tkinter import ttk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.boton1=ttk.Button(self.ventana1, text="Boton 1")

        self.boton1.grid(column=0, row=0)

        self.boton2=ttk.Button(self.ventana1, text="Boton 2")

        self.boton2.grid(column=1, row=0)

        self.boton3=ttk.Button(self.ventana1, text="Boton 3")

        self.boton3.grid(column=2, row=0, rowspan=2, sticky="ns")

        self.boton4=ttk.Button(self.ventana1, text="Boton 4")

        self.boton4.grid(column=0, row=1)

        self.boton5=ttk.Button(self.ventana1, text="Boton 5")

        self.boton5.grid(column=1, row=1)

        self.boton6=ttk.Button(self.ventana1, text="Boton 6")

        self.boton6.grid(column=0, row=2, columnspan=3, sticky="we")

        self.ventana1.mainloop()

aplicacion1=Aplicacion()

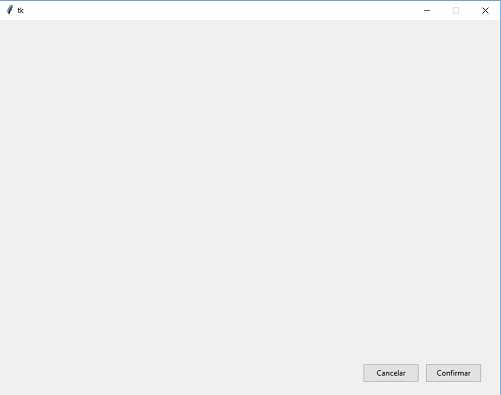
### Layout Manager: Place

Este tipo de Layout Manager nos permite disponer un Widget en una posición y con un tamaño con valor absoluto a nivel de píxeles. Hay que tener cuidado en que casos utilizar este tipo de administrador de diseños ya que si agrandamos o reducimos el tamaño de la ventana puede ser que los controles queden fuera de la ventana y el operador no pueda visualizarlos.

### Problema:

Disponer dos botones en la parte inferior derecha de la ventana utilizando el Layout Manager de tipo Place. El ancho y alto de la ventana debe ser de 800 por 600 píxeles.

La representación visual debe ser:



#### Programa: ejercicio248.py

import tkinter as tk

from tkinter import ttk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.ventana1.geometry("800x600")

        self.ventana1.resizable(0,0)

        self.boton1=ttk.Button(self.ventana1, text="Confirmar")

        self.boton1.place(x=680, y=550, width=90, height=30)

        self.boton2=ttk.Button(self.ventana1, text="Cancelar")

        self.boton2.place(x=580, y=550, width=90, height=30)

        self.ventana1.mainloop()

aplicacion1=Aplicacion()

### Recomendación

Siempre que tenga que implementar una interfaz gráfica es conveniente que en un papel se haga un croquis y a partir de este agrupar controles relacionados dentro de Frame o LabelFrame.

tkinter.messagebox : ventanas de mensajes

Las aplicaciones de escritorio es muy común la necesidad de abrir otras ventanas emergentes con el objetivo de informar, advertir de errores etc.

La librería tkinter provee un paquete llamado messagebox con una serie de funciones para la apertura de diálogos de información.

Para usar estos diálogos lo primero que debemos hacer es importar el paquete:

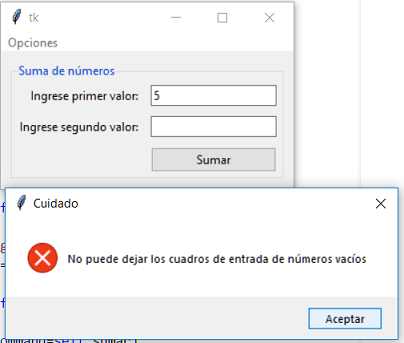
from tkinter import messagebox as mb

Podemos crear un alias para el nombre del paquete para no tener que escribir messagebox todas las veces.

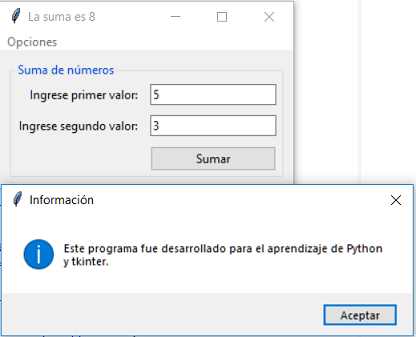
Problema:

Confeccionar una aplicación que permita ingresar dos valores enteros y al presionar un botón nos muestre la suma en el título de la ventana. Si el operador no ingresa en alguno de los dos controles Entry datos informar mediante un diálogo el error que se está cometiendo.  
Agregar además un menú de opciones que al ser seleccionado nos muestre información del programa.

La representación visual cuando se presiona el botón "sumar" y no se ingresa alguno de los dos números debe ser:



La representación visual cuando se abre el diálogo que informa sobre el programa desde el menú de opciones es:



Programa: ejercicio249.py

import tkinter as tk

from tkinter import ttk

from tkinter import messagebox as mb

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.labelframe1=ttk.LabelFrame(self.ventana1, text="Suma de números")

        self.labelframe1.grid(column=0, row=0, padx=10, pady=10)

        self.agregar\_componentes()

        self.agregar\_menu()

        self.ventana1.mainloop()

    def agregar\_componentes(self):

        self.label1=ttk.Label(self.labelframe1, text="Ingrese primer valor:")

        self.label1.grid(column=0, row=0, padx=5, pady=5, sticky="e")

        self.dato1=tk.StringVar()

        self.entry1=ttk.Entry(self.labelframe1, textvariable=self.dato1)

        self.entry1.grid(column=1, row=0, padx=5, pady=5)

        self.label2=ttk.Label(self.labelframe1, text="Ingrese segundo valor:")

        self.label2.grid(column=0, row=1, padx=5, pady=5, sticky="e")

        self.dato2=tk.StringVar()

        self.entry2=ttk.Entry(self.labelframe1, textvariable=self.dato2)

        self.entry2.grid(column=1, row=1, padx=5, pady=5)

        self.boton1=ttk.Button(self.labelframe1, text="Sumar", command=self.sumar)

        self.boton1.grid(column=1, row=2, padx=5, pady=5, sticky="we")

    def agregar\_menu(self):

        self.menubar1 = tk.Menu(self.ventana1)

        self.ventana1.config(menu=self.menubar1)

        self.opciones1 = tk.Menu(self.menubar1, tearoff=0)

        self.opciones1.add\_command(label="Acerca de...", command=self.acerca)

        self.menubar1.add\_cascade(label="Opciones", menu=self.opciones1)

    def sumar(self):

        if self.dato1.get()=="" or self.dato2.get()=="":

            mb.showerror("Cuidado","No puede dejar los cuadros de entrada de números vacíos")

        else:

            suma=int(self.dato1.get())+int(self.dato2.get())

            self.ventana1.title("La suma es "+str(suma))

    def acerca(self):

        mb.showinfo("Información", "Este programa fue desarrollado para el aprendizaje de Python y tkinter.")

aplicacion1=Aplicacion()

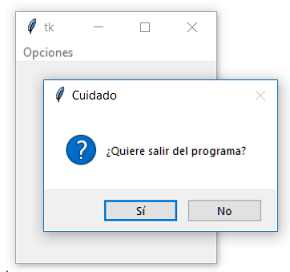
Diálogos para confirmar o rechazar.

El paquete 'messagebox' cuenta con otra función que nos muestra un diálogo con dos botones con los mensajes "Si" o "No", luego desde nuestro programa podemos identificar cual de los dos botones se ha presionado.

### Problema:

Confeccionar un programa que tenga solo un menú de opciones que al ser presionado nos muestre un cuadro de mensaje que informe si queremos finalizar la ejecución del programa. Si se presiona "si" se finaliza el programa en caso contrario no se hace nada.

La representación visual luego de seleccionar la opción del menú es:



#### Programa: ejercicio250.py

import tkinter as tk

from tkinter import messagebox as mb

import sys

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.agregar\_menu()

        self.ventana1.mainloop()

    def agregar\_menu(self):

        self.menubar1 = tk.Menu(self.ventana1)

        self.ventana1.config(menu=self.menubar1)

        self.opciones1 = tk.Menu(self.menubar1, tearoff=0)

        self.opciones1.add\_command(label="Salir", command=self.salir)

        self.menubar1.add\_cascade(label="Opciones", menu=self.opciones1)

    def salir(self):

        respuesta=mb.askyesno("Cuidado", "¿Quiere salir del programa?")

        if respuesta==True:

            sys.exit()

aplicacion1=Aplicacion()

# ventanas de diálogos

Vimos en el concepto anterior que tkinter nos provee una serie de diálogos para mostrar mensajes de información y error. En muchas situaciones podemos necesitar crear otras ventanas con más funcionalidades que los simples ventanas de mensajes.

Para crear diálogos en tkinter debemos crear un objeto de la clase TopLevel y pasar como parámetro la referencia de la ventana principal.

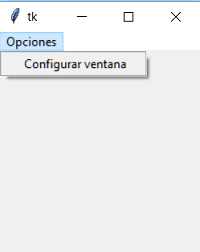
Un diálogo se asemeja mucho a lo que es la ventana principal, podemos disponer dentro de la misma objetos de la clase Label, Button, Entry etc.

Lo más común es que un diálogo tenga por objetivo la entrada de datos para luego ser utilizados en la ventana principal.

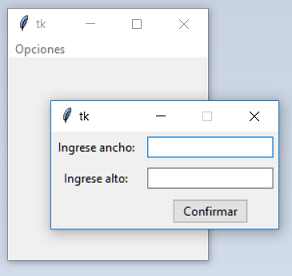
### Problema:

Confeccionar una aplicación que muestre un diálogo cuando se seleccione una opción de un menú.  
El dialogo debe solicitar el ingreso de dos enteros que se utilizarán en la ventana principal para redimensionarla.

La representación visual de la ventana principal es muy sencilla:



Cuando se abre el diálogo tenemos la siguiente imagen:



#### Programa: ejercicio251.py

import tkinter as tk

from tkinter import ttk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.agregar\_menu()

        self.ventana1.mainloop()

    def agregar\_menu(self):

        self.menubar1 = tk.Menu(self.ventana1)

        self.ventana1.config(menu=self.menubar1)

        self.opciones1 = tk.Menu(self.menubar1, tearoff=0)

        self.opciones1.add\_command(label="Configurar ventana", command=self.configurar)

        self.menubar1.add\_cascade(label="Opciones", menu=self.opciones1)

    def configurar(self):

        dialogo1 = DialogoTamano(self.ventana1)

        s=dialogo1.mostrar()

        self.ventana1.geometry(s[0]+"x"+s[1])

class DialogoTamano:

    def \_\_init\_\_(self, ventanaprincipal):

        self.dialogo=tk.Toplevel(ventanaprincipal)

        self.label1=ttk.Label(self.dialogo, text="Ingrese ancho:")

        self.label1.grid(column=0, row=0, padx=5, pady=5)

        self.dato1=tk.StringVar()

        self.entry1=ttk.Entry(self.dialogo, textvariable=self.dato1)

        self.entry1.grid(column=1, row=0, padx=5, pady=5)

        self.entry1.focus()

        self.label2=ttk.Label(self.dialogo, text="Ingrese alto:")

        self.label2.grid(column=0, row=1, padx=5, pady=5)

        self.dato2=tk.StringVar()

        self.entry2=ttk.Entry(self.dialogo, textvariable=self.dato2)

        self.entry2.grid(column=1, row=1, padx=5, pady=5)

        self.boton1=ttk.Button(self.dialogo, text="Confirmar", command=self.confirmar)

        self.boton1.grid(column=1, row=2, padx=5, pady=5)

        self.dialogo.protocol("WM\_DELETE\_WINDOW", self.confirmar)

        self.dialogo.resizable(0,0)

        self.dialogo.grab\_set()

    def mostrar(self):

        self.dialogo.wait\_window()

        return (self.dato1.get(), self.dato2.get())

    def confirmar(self):

        self.dialogo.destroy()

aplicacion1=Aplicacion()

ttk : control Spinbox

Este control visual es muy utilizado para seleccionar un valor de una lista. Se dispone de dos botones para subir o bajar en la lista de valores posibles:

Spinbox

Se pueden mostrar listas de valores de tipo string.

Problema:

En una aduana hay una máquina que sortea las personas cuyo equipaje serán revisados.  
La persona selecciona la cantidad de bultos (hacer dicha selección mediante un Spinbox)

Luego se presiona el botón sortear y aparece al lado de este botón una Label de color rojo o verde (En caso de ser rojo se revisa su equipaje, en caso de ser verde, no se revisa)  
Para el sorteo generar un valor aleatorio entre 1 y 3. Si se genera un 1 se revisa, si se genera un 2 o 3 no se revisa, mostrar un mensaje de error si el Spinbox tiene un cero.

La interfaz visual debe ser similar a esta:



Programa: ejercicio252.py

import tkinter as tk

from tkinter import ttk

from tkinter import messagebox as mb

import random

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.label1=ttk.Label(self.ventana1, text="Seleccione la cantidad de bultos:")

        self.label1.grid(column=0, row=0, padx=10, pady=10)

        self.spinbox1=ttk.Spinbox(self.ventana1, from\_=0, to=100, width=10, state='readonly')

        self.spinbox1.set(0)

        self.spinbox1.grid(column=1, row=0, padx=10, pady=10)

        self.boton1=ttk.Button(self.ventana1, text="Sortear", command=self.sortear)

        self.boton1.grid(column=0, row=1, padx=10, pady=10)

        self.label2=ttk.Label(self.ventana1, text="", width=20)

        self.label2.grid(column=1, row=1, padx=10, pady=10)

        self.ventana1.mainloop()

    def sortear(self):

        if int(self.spinbox1.get())==0:

            mb.showerror("Cuidado","Debe seleccionar un valor distinto a cero en bultos")

        else:

            valor=random.randint(1,3)

            if valor==1:

                self.label2.configure(text="Se deben revisar")

                self.label2.configure(background="red")

            else:

                self.label2.configure(text="No se revisan")

                self.label2.configure(background="green")

aplicacion1=Aplicacion()

### Acotaciones

* Si queremos llenar el Spinbox con valores no consecutivos, sino de 3 en 3 podemos utilizar la siguiente sintaxis:

self.spinbox1=ttk.Spinbox(self.ventana1, **increment=3**, from\_=1, to=10, state='readonly')

* Podemos definir una tupla y luego inicializar el parámetro values:

dias=("lunes", "martes", "miércoles","jueves","viernes","sábado","domingo")

self.spinbox1=ttk.Spinbox(self.ventana1, values=dias, state='readonly')

self.spinbox1.set(dias[0])

# tkinter.ScrolledText : editor multilínea

Hemos visto anteriormente el control Entry que nos permite ingresar una cadena de caracteres por teclado. El Widget ScrolledText es similar al Widget Entry con la salvedad que nos permite ingresar múltiples líneas.

Cuenta con métodos sofisticados para extraer trozos del texto ingresado en la componente.

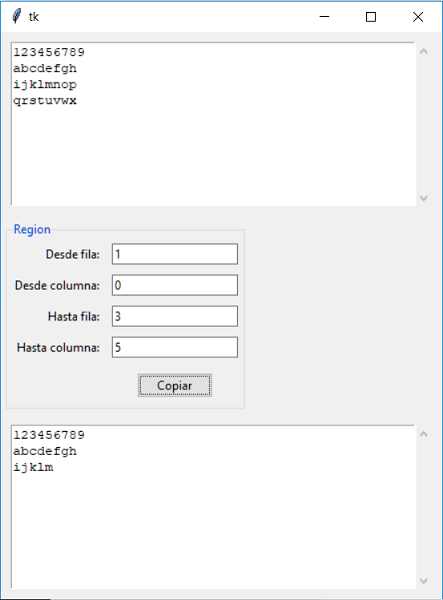
Para poder utilizar esta nueva componente debemos importar el módulo scrolledtext del paquete tkinter (podemos crear un alias para no tener que escribir scrolledtext):

from tkinter import scrolledtext as st

### Problema:

Confeccionar un programa que contenga dos controles de tipo ScrolledText. En el primero ingresamos por teclado cualquier texto. Mediante 4 controles de tipo Entry indicar desde que fila y columna hasta que fila y columna extraer caracteres del primer ScrolledText y copiarlos al segundo ScrolledText cuando se presione un botón.

La interfaz visual debe ser similar a esta:



#### Programa: ejercicio253.py

import tkinter as tk

from tkinter import ttk

from tkinter import scrolledtext as st

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.scrolledtext1=st.ScrolledText(self.ventana1, width=50, height=10)

        self.scrolledtext1.grid(column=0,row=0, padx=10, pady=10)

        self.framecopia()

        self.scrolledtext2=st.ScrolledText(self.ventana1, width=50, height=10)

        self.scrolledtext2.grid(column=0,row=2, padx=10, pady=10)

        self.ventana1.mainloop()

    def framecopia(self):

        self.labelframe1=ttk.LabelFrame(self.ventana1, text="Region")

        self.labelframe1.grid(column=0, row=1, padx=5, pady=5, sticky="w")

        self.label1=ttk.Label(self.labelframe1, text="Desde fila:")

        self.label1.grid(column=0, row=0, padx=5, pady=5, sticky="e")

        self.dato1=tk.StringVar()

        self.entry1=ttk.Entry(self.labelframe1, textvariable=self.dato1)

        self.entry1.grid(column=1, row=0, padx=5, pady=5, sticky="e")

        self.label2=ttk.Label(self.labelframe1, text="Desde columna:")

        self.label2.grid(column=0, row=1, padx=5, pady=5, sticky="e")

        self.dato2=tk.StringVar()

        self.entry2=ttk.Entry(self.labelframe1, textvariable=self.dato2)

        self.entry2.grid(column=1, row=1, padx=5, pady=5, sticky="e")

        self.label3=ttk.Label(self.labelframe1, text="Hasta fila:")

        self.label3.grid(column=0, row=2, padx=5, pady=5, sticky="e")

        self.dato3=tk.StringVar()

        self.entry3=ttk.Entry(self.labelframe1, textvariable=self.dato3)

        self.entry3.grid(column=1, row=2, padx=5, pady=5, sticky="e")

        self.label4=ttk.Label(self.labelframe1, text="Hasta columna:")

        self.label4.grid(column=0, row=3, padx=5, pady=5, sticky="e")

        self.dato4=tk.StringVar()

        self.entry4=ttk.Entry(self.labelframe1, textvariable=self.dato4)

        self.entry4.grid(column=1, row=3, padx=5, pady=5, sticky="e")

        self.boton1=ttk.Button(self.labelframe1, text="Copiar", command=self.copiar)

        self.boton1.grid(column=1, row=4, padx=10, pady=10)

    def copiar(self):

        iniciofila=self.dato1.get()

        iniciocolumna=self.dato2.get()

        finfila=self.dato3.get()

        fincolumna=self.dato4.get()

        datos=self.scrolledtext1.get(iniciofila+"."+iniciocolumna, finfila+"."+fincolumna)

        self.scrolledtext2.delete("1.0", tk.END)

        self.scrolledtext2.insert("1.0", datos)

aplicacion1=Aplicacion()

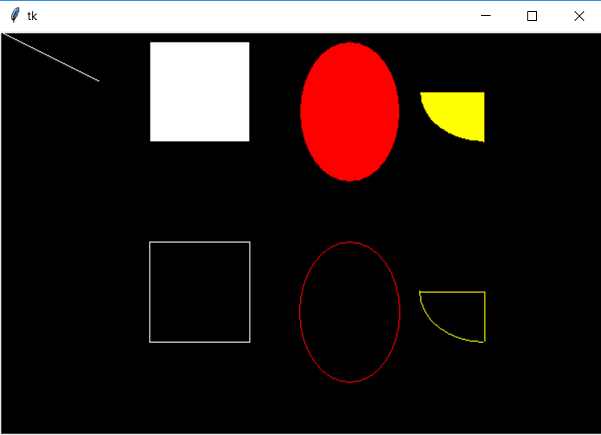
# tkinter: lienzo de control

El lienzo de control nos permite acceder a una serie de gráficas primitivas: líneas, rectángulos, óvalos, arcos, etc. para graficar dentro de la misma.

### Problema:

Confeccione un programa que crea un objeto de la clase Canvas y muestre la funcionalidad de las principales gráficas primitivas.

La interfaz visual debe ser similar a esta:



#### Programa: ejercicio254.py

import tkinter as tk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.canvas1=tk.Canvas(self.ventana1, width=600, height=400, background="black")

        self.canvas1.grid(column=0, row=0)

        self.canvas1.create\_line(0, 0, 100,50, fill="white")

        self.canvas1.create\_rectangle(150,10, 250,110, fill="white")

        self.canvas1.create\_oval(300,10,400,150, fill="red")

        self.canvas1.create\_arc(420,10,550,110, fill="yellow", start=180, extent=90)

        self.canvas1.create\_rectangle(150,210, 250,310, outline="white")

        self.canvas1.create\_oval(300,210,400,350, outline="red")

        self.canvas1.create\_arc(420,210,550,310, outline="yellow", start=180, extent=90)

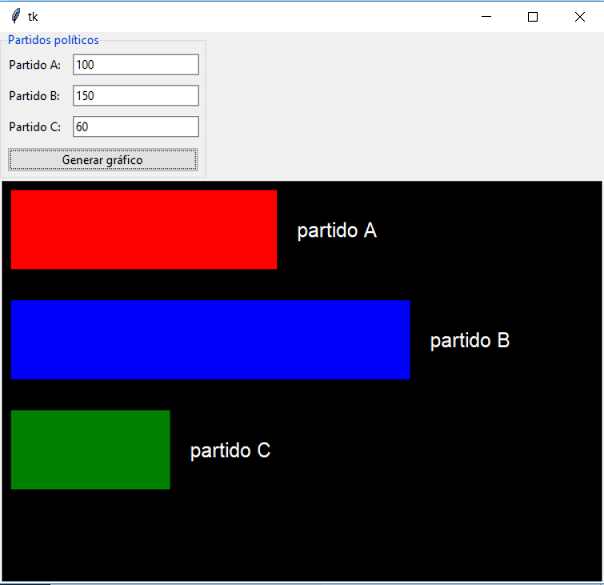
        self.ventana1.mainloop()

aplicacion1=Aplicacion(

### Problema:

Crear una aplicación que solicite el ingreso de tres valores por teclado que representan las cantidades de votos obtenidas por tres partidos políticos. Luego mostrar un gráfico de barras horizontales.

La interfaz visual debe ser similar a esta:



#### Programa: ejercicio255.py

import tkinter as tk

from tkinter import ttk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.entradadatos()

        self.canvas1=tk.Canvas(self.ventana1, width=600, height=400, background="black")

        self.canvas1.grid(column=0, row=1)

        self.ventana1.mainloop()

    def entradadatos(self):

        self.lf1=ttk.LabelFrame(self.ventana1,text="Partidos políticos")

        self.lf1.grid(column=0, row=0, sticky="w")

        self.label1=ttk.Label(self.lf1, text="Partido A:")

        self.label1.grid(column=0,row=0, padx=5, pady=5)

        self.dato1=tk.StringVar()

        self.entry1=ttk.Entry(self.lf1, textvariable=self.dato1)

        self.entry1.grid(column=1, row=0, padx=5, pady=5)

        self.label2=ttk.Label(self.lf1, text="Partido B:")

        self.label2.grid(column=0,row=1, padx=5, pady=5)

        self.dato2=tk.StringVar()

        self.entry2=ttk.Entry(self.lf1, textvariable=self.dato2)

        self.entry2.grid(column=1, row=1, padx=5, pady=5)

        self.label3=ttk.Label(self.lf1, text="Partido C:")

        self.label3.grid(column=0,row=2, padx=5, pady=5)

        self.dato3=tk.StringVar()

        self.entry3=ttk.Entry(self.lf1, textvariable=self.dato3)

        self.entry3.grid(column=1, row=2, padx=5, pady=5)

        self.boton1=ttk.Button(self.lf1, text="Generar gráfico", command=self.grafico\_barra)

        self.boton1.grid(column=0, row=3, columnspan=2, padx=5, pady=5, sticky="we")

        self.entry1.focus()

    def grafico\_barra(self):

        self.canvas1.delete(tk.ALL)

        valor1=int(self.dato1.get())

        valor2=int(self.dato2.get())

        valor3=int(self.dato3.get())

        if valor1>valor2 and valor1>valor3:

            mayor=valor1

        else:

            if valor2>valor3:

                mayor=valor2

            else:

                mayor=valor3

        largo1=valor1/mayor\*400

        largo2=valor2/mayor\*400

        largo3=valor3/mayor\*400

        self.canvas1.create\_rectangle(10,10,10+largo1,90,fill="red")

        self.canvas1.create\_rectangle(10,120,10+largo2,200,fill="blue")

        self.canvas1.create\_rectangle(10,230,10+largo3,310,fill="green")

        self.canvas1.create\_text(largo1+70, 50, text="partido A", fill="white", font="Arial")

        self.canvas1.create\_text(largo2+70, 160, text="partido B", fill="white", font="Arial")

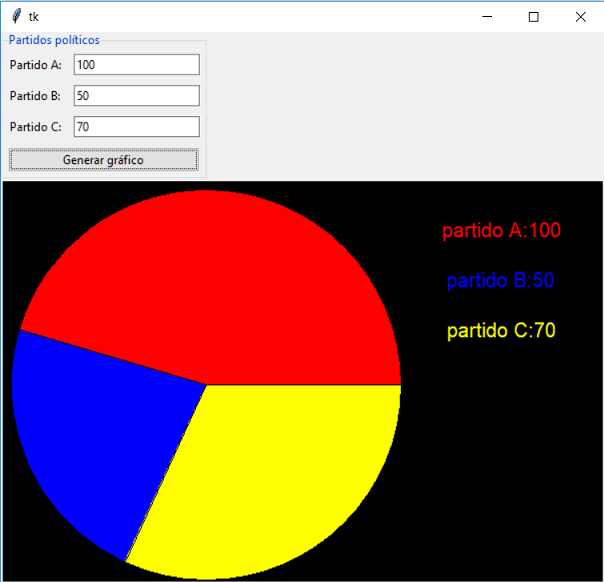
        self.canvas1.create\_text(largo3+70, 270, text="partido C", fill="white", font="Arial")

aplicacion1=Aplicacion()

### Problema:

Crear una aplicación que solicite el ingreso de tres valores por teclado que representan las cantidades de votos obtenidas por tres partidos políticos. Luego mostrar un gráfico de tartas:.

La interfaz visual debe ser similar a esta:



#### Programa: ejercicio256.py

import tkinter as tk

from tkinter import ttk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.entradadatos()

        self.canvas1=tk.Canvas(self.ventana1, width=600, height=400, background="black")

        self.canvas1.grid(column=0, row=1)

        self.ventana1.mainloop()

    def entradadatos(self):

        self.lf1=ttk.LabelFrame(self.ventana1,text="Partidos políticos")

        self.lf1.grid(column=0, row=0, sticky="w")

        self.label1=ttk.Label(self.lf1, text="Partido A:")

        self.label1.grid(column=0,row=0, padx=5, pady=5)

        self.dato1=tk.StringVar()

        self.entry1=ttk.Entry(self.lf1, textvariable=self.dato1)

        self.entry1.grid(column=1, row=0, padx=5, pady=5)

        self.label2=ttk.Label(self.lf1, text="Partido B:")

        self.label2.grid(column=0,row=1, padx=5, pady=5)

        self.dato2=tk.StringVar()

        self.entry2=ttk.Entry(self.lf1, textvariable=self.dato2)

        self.entry2.grid(column=1, row=1, padx=5, pady=5)

        self.label3=ttk.Label(self.lf1, text="Partido C:")

        self.label3.grid(column=0,row=2, padx=5, pady=5)

        self.dato3=tk.StringVar()

        self.entry3=ttk.Entry(self.lf1, textvariable=self.dato3)

        self.entry3.grid(column=1, row=2, padx=5, pady=5)

        self.boton1=ttk.Button(self.lf1, text="Generar gráfico", command=self.grafico\_tarta)

        self.boton1.grid(column=0, row=3, columnspan=2, padx=5, pady=5, sticky="we")

        self.entry1.focus()

    def grafico\_tarta(self):

        self.canvas1.delete(tk.ALL)

        valor1=int(self.dato1.get())

        valor2=int(self.dato2.get())

        valor3=int(self.dato3.get())

        suma=valor1+valor2+valor3

        grados1=valor1/suma\*360

        grados2=valor2/suma\*360

        grados3=valor3/suma\*360

        self.canvas1.create\_arc(10,10,400,400,fill="red", start=0, extent=grados1)

        self.canvas1.create\_arc(10,10,400,400,fill="blue", start=grados1, extent=grados2)

        self.canvas1.create\_arc(10,10,400,400,fill="yellow", start=grados1+grados2, extent=grados3)

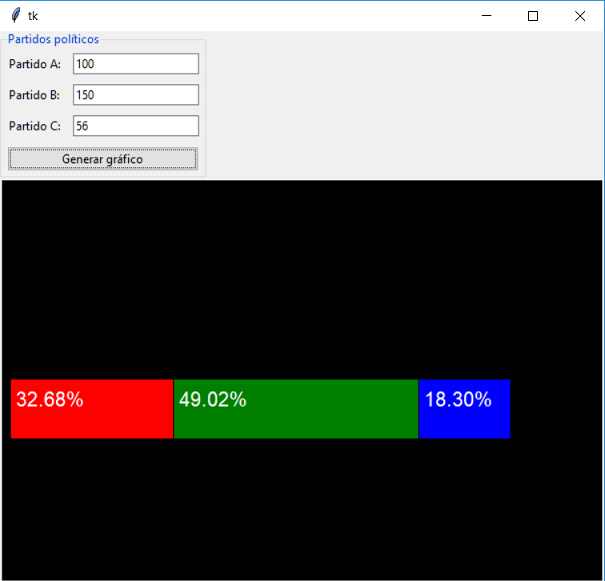
        self.canvas1.create\_text(500, 50, text="partido A:"+str(valor1), fill="red", font="Arial")

        self.canvas1.create\_text(500, 100, text="partido B:"+str(valor2), fill="blue", font="Arial")

        self.canvas1.create\_text(500, 150, text="partido C:"+str(valor3), fill="yellow", font="Arial")

aplicacion1=Aplicacion()

Problema propuesto

* Implementar un gráfico estadístico de tipo "Barra Porcentual":  
  
* ejercicio257.py
* import tkinter as tk
* from tkinter import ttk
* class Aplicacion:
* def \_\_init\_\_(self):
* self.ventana1=tk.Tk()
* self.entradadatos()
* self.canvas1=tk.Canvas(self.ventana1, width=600, height=400, background="black")
* self.canvas1.grid(column=0, row=1)
* self.ventana1.mainloop()
* def entradadatos(self):
* self.lf1=ttk.LabelFrame(self.ventana1,text="Partidos políticos")
* self.lf1.grid(column=0, row=0, sticky="w")
* self.label1=ttk.Label(self.lf1, text="Partido A:")
* self.label1.grid(column=0,row=0, padx=5, pady=5)
* self.dato1=tk.StringVar()
* self.entry1=ttk.Entry(self.lf1, textvariable=self.dato1)
* self.entry1.grid(column=1, row=0, padx=5, pady=5)
* self.label2=ttk.Label(self.lf1, text="Partido B:")
* self.label2.grid(column=0,row=1, padx=5, pady=5)
* self.dato2=tk.StringVar()
* self.entry2=ttk.Entry(self.lf1, textvariable=self.dato2)
* self.entry2.grid(column=1, row=1, padx=5, pady=5)
* self.label3=ttk.Label(self.lf1, text="Partido C:")
* self.label3.grid(column=0,row=2, padx=5, pady=5)
* self.dato3=tk.StringVar()
* self.entry3=ttk.Entry(self.lf1, textvariable=self.dato3)
* self.entry3.grid(column=1, row=2, padx=5, pady=5)
* self.boton1=ttk.Button(self.lf1, text="Generar gráfico", command=self.grafico\_barraporcentual)
* self.boton1.grid(column=0, row=3, columnspan=2, padx=5, pady=5, sticky="we")
* self.entry1.focus()
* def grafico\_barraporcentual(self):
* self.canvas1.delete(tk.ALL)
* valor1=int(self.dato1.get())
* valor2=int(self.dato2.get())
* valor3=int(self.dato3.get())
* suma=valor1+valor2+valor3
* largo1=valor1\*500/suma
* largo2=valor2\*500/suma
* largo3=valor3\*500/suma
* porc1=valor1/suma\*100
* porc2=valor2/suma\*100
* porc3=valor3/suma\*100
* self.canvas1.create\_rectangle(10,200,10+largo1,260,fill="red")
* self.canvas1.create\_text(50, 220, text="{0:.2f}".format(porc1)+"%", fill="white", font="Arial")
* self.canvas1.create\_rectangle(10+largo1,200,10+largo1+largo2,260,fill="green")
* self.canvas1.create\_text(50+largo1, 220, text="{0:.2f}".format(porc2)+"%", fill="white", font="Arial")
* self.canvas1.create\_rectangle(10+largo1+largo2,200,10+largo1+largo2+largo3,260,fill="blue")
* self.canvas1.create\_text(50+largo1+largo2, 220, text="{0:.2f}".format(porc3)+"%", fill="white", font="Arial")
* aplicacion1=Aplicacion()

# Canvas : captura de eventos del mouse

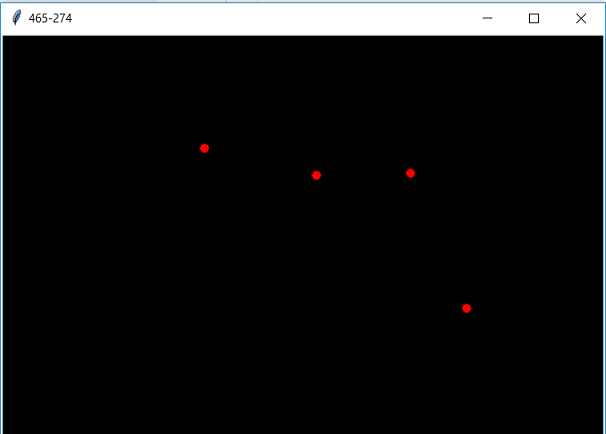
Una actividad muy común es poder detectar cuando hemos presionado el mouse dentro de un control Canvas (o de cualquier otro Widget que hayamos visto) y a partir de eso disparar un evento.

Veremos con una serie de ejemplos la posibilidades de tkinter nos provee.

### Problema:

Confeccionar un programa que cree un objeto de la clase Canvas y nos muestre en el título de la ventana la coordenada actual del mouse dentro del control Canvas y al presionar el botón izquierdo del mouse se dibuje un círculo en dicha posición.

La interfaz visual debe ser similar a esta:



#### Programa: ejercicio258.py

import tkinter as tk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.canvas1=tk.Canvas(self.ventana1, width=600, height=400, background="black")

        self.canvas1.bind("<Motion>", self.mover\_mouse)

        self.canvas1.bind("<Button-1>", self.presion\_mouse)

        self.canvas1.grid(column=0, row=1)

        self.ventana1.mainloop()

    def presion\_mouse(self, evento):

        self.canvas1.create\_oval(evento.x-5,evento.y-5,evento.x+5,evento.y+5, fill="red")

    def mover\_mouse(self, evento):

        self.ventana1.title(str(evento.x)+"-"+str(evento.y))

aplicacion1=Aplicacion()

### Acotaciones

La diversidad de eventos que podemos capturar es muy grande, veamos algunos ejemplos:

* Para capturar el evento clic del botón izquierdo del mouse indicamos en el método bind <Button-1>, el botón central <Button-2> y el botón derecho del mouse <Button-3>
* Si necesitamos capturar el evento clic del botón derecho del mouse y a su vez que se encuentre presionada la tecla Shift tenemos que codificar:
* self.canvas1.bind("<Shift Button-1>", self.presion\_mouse)

En lugar de Shift podemos verificar si se está presionando la tecla control:

self.canvas1.bind("<Control Button-1>", self.presion\_mouse)

Inclusive detectar el evento si se presiona Shift, Control y el botón izquierdo del mouse:

self.canvas1.bind("<Control Shift Button-1>", self.presion\_mouse)

La tecla Alt, Shift, Control y el botón izquierdo del mouse:

self.canvas1.bind("<Control Shift Alt Button-1>", self.presion\_mouse)

* Si necesitamos hacer algo cuando la flecha del mouse entra al control podemos plantear la captura del evento:

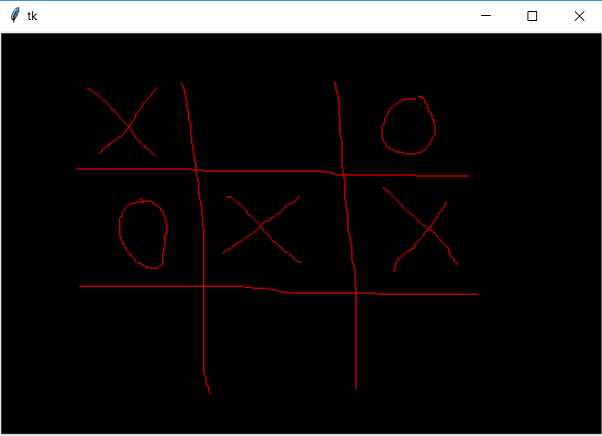
self.canvas1.bind("<Enter>", self.entrada)

Y si queremos detectar cuando la flecha del mouse sale de la componente:

self.canvas1.bind("<Leave>", self.salida)

* Para detectar el doble clic de un botón del mouse:

self.canvas1.bind("<Double-Button-1>", self.presion\_mouse)



import tkinter as tk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.canvas1=tk.Canvas(self.ventana1, width=600, height=400, background="black")

        self.canvas1.grid(column=0,row=0)

        self.canvas1.bind("<ButtonPress-1>",self.boton\_presion)

        self.canvas1.bind("<Motion>", self.mover\_mouse)

        self.canvas1.bind("<ButtonRelease-1>",self.boton\_soltar)

        self.presionado=False

        self.ventana1.mainloop()

    def boton\_presion(self, evento):

        self.presionado=True

        self.origenx=evento.x

        self.origeny=evento.y

    def mover\_mouse(self, evento):

        if self.presionado:

            self.canvas1.create\_line(self.origenx,self.origeny,evento.x,evento.y, fill="red")

            self.origenx=evento.x

            self.origeny=evento.y

    def boton\_soltar(self,evento):

        self.presionado=False

aplicacion1=Aplicacion()

Canvas : borrar figuras mediante Ids y Tags

Cuando creamos una figura con los métodos que provee la clase Canvas, el mismo nos retorna un Id (identificador) de dicha figura. Luego podemos hacer referencia a la misma mediante ese Id.

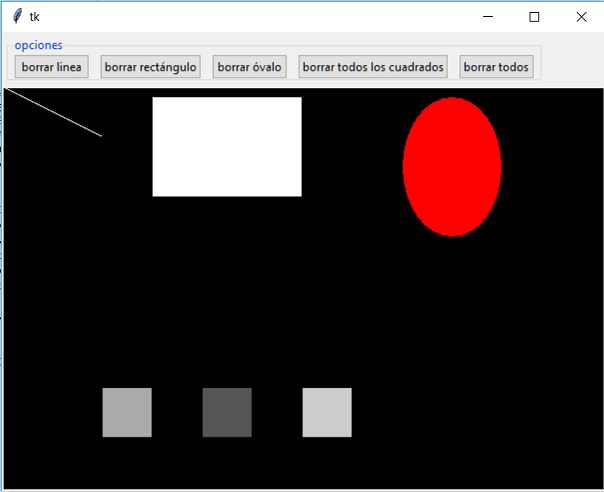
Podemos asociar además a una o más figuras un Tag (marca) y luego borrar todas las figuras que tienen dicho Tag.

### Problema:

Confeccionar un programa que cree un objeto de la clase Canvas y luego dibuje una línea, un réctángulo y un óvalo, almacenar el Id de cada dibujo en un atributo.  
Crear también 3 cuadrados y definir el parámetro tag con el mismo valor para cada uno de ellos.

Mediante cinco botones permitir: borrar la línea, borrar el rectángulo, borrar el óvalo, borrar todos los cuadrados y borrar todas las figuras contenidas en el objeto Canvas.

La interfaz visual debe ser similar a esta:



#### Programa: ejercicio260.py

import tkinter as tk

from tkinter import ttk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.crear\_botones()

        self.canvas1=tk.Canvas(self.ventana1, width=600, height=400, background="black")

        self.canvas1.grid(column=0, row=1)

        self.linea=self.canvas1.create\_line(0, 0, 100,50, fill="white")

        self.rectangulo=self.canvas1.create\_rectangle(150,10, 300,110, fill="white")

        self.ovalo=self.canvas1.create\_oval(400,10,500,150, fill="red")

        self.canvas1.create\_rectangle(100,300,150,350,fill="#aaaaaa", tag="cuadrado")

        self.canvas1.create\_rectangle(200,300,250,350,fill="#555555", tag="cuadrado")

        self.canvas1.create\_rectangle(300,300,350,350,fill="#cccccc", tag="cuadrado")

        self.ventana1.mainloop()

    def crear\_botones(self):

        self.labelframe1=ttk.LabelFrame(self.ventana1,text="opciones")

        self.labelframe1.grid(column=0, row=0, sticky="w", padx=5, pady=5)

        self.boton1=ttk.Button(self.labelframe1, text="borrar linea", command=self.borrar\_linea)

        self.boton1.grid(column=0, row=0, padx=5)

        self.boton2=ttk.Button(self.labelframe1, text="borrar rectángulo", command=self.borrar\_rectangulo)

        self.boton2.grid(column=1, row=0, padx=5)

        self.boton3=ttk.Button(self.labelframe1, text="borrar óvalo", command=self.borrar\_ovalo)

        self.boton3.grid(column=2, row=0, padx=5)

        self.boton4=ttk.Button(self.labelframe1, text="borrar todos los cuadrados", command=self.borrar\_cuadrados)

        self.boton4.grid(column=3, row=0, padx=5)

        self.boton5=ttk.Button(self.labelframe1, text="borrar todos", command=self.borrar\_todos)

        self.boton5.grid(column=4, row=0, padx=5)

    def borrar\_linea(self):

        self.canvas1.delete(self.linea)

    def borrar\_rectangulo(self):

        self.canvas1.delete(self.rectangulo)

    def borrar\_ovalo(self):

        self.canvas1.delete(self.ovalo)

    def borrar\_cuadrados(self):

        self.canvas1.delete("cuadrado")

    def borrar\_todos(self):

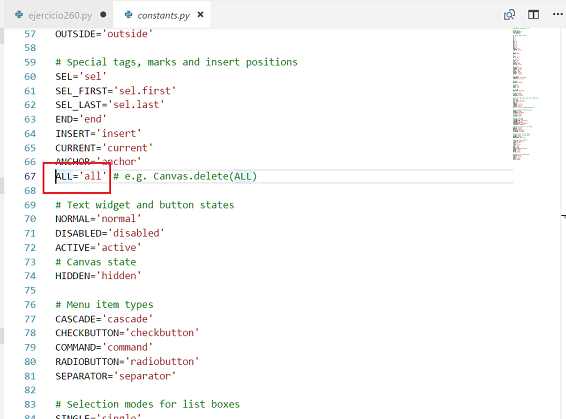
        self.canvas1.delete(tk.ALL)

aplicacion1=Aplicacion()

### Acotaciones

Cuando trabajamos con el editor Visual Studio Code podemos navegar la referencia de variables, métodos, clases etc. presionando la tecla Control y posicionando la flecha del mouse sobre el identificar:



Si hacemos clic luego que aparece subrayado se abre el archivo que contiene dicha definición: 

Esta posibilidad de navegar el código fuente de los módulos nos puede facilitar entender su funcionamiento. Según lo visto la variable ALL del módulo tkinter almacena la cadena 'all', luego podemos escribir la eliminación de todos los gráficos con la sintaxis:

def borrar\_todos(self):

self.canvas1.delete('all')

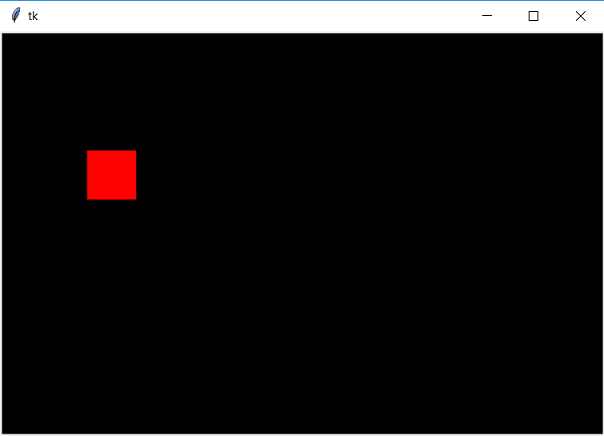
# Canvas : desplazar una figura mediante el método move

Otra situación donde tiene sentido almacenar el Id de una figura es cuando en un momento posterior queremos desplazarla a otra posición dentro del objeto Canvas.

Problema:

Confeccionar un programa que que muestre un cuadrado dentro de un Canvas. Cuando se presione alguna de las teclas de flecha proceder a desplazar la figura 4 píxeles teniendo en cuenta la dirección de la tecla de flecha presionada.

La interfaz visual debe ser similar a esta:



Programa: ejercicio261.py

import tkinter as tk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.canvas1=tk.Canvas(self.ventana1, width=600, height=400, background="black")

        self.canvas1.grid(column=0, row=0)

        self.cuadrado=self.canvas1.create\_rectangle(150,10,200,60, fill="red")

        self.ventana1.bind("<KeyPress>", self.presion\_tecla)

        self.ventana1.mainloop()

    def presion\_tecla(self, evento):

        if evento.keysym=='Right':

            self.canvas1.move(self.cuadrado, 4, 0)

        if evento.keysym=='Left':

            self.canvas1.move(self.cuadrado, -4, 0)

        if evento.keysym=='Down':

            self.canvas1.move(self.cuadrado, 0, 4)

        if evento.keysym=='Up':

            self.canvas1.move(self.cuadrado, 0, -4)

aplicacion1=Aplicacion()

Problema propuesto

* Modificar el problema que desplaza un cuadrado con las teclas de flechas de tal modo que la figura no pueda salir del espacio definido para el Canvas.  
  Para saber la posición actual de una figura la clase Canvas cuenta con el método 'coords':

ejercicio262.py

import tkinter as tk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.canvas1=tk.Canvas(self.ventana1, width=600, height=400, background="black")

        self.canvas1.grid(column=0, row=0)

        self.cuadrado=self.canvas1.create\_rectangle(150,10,200,60, fill="red")

        self.ventana1.bind("<KeyPress>", self.presion\_tecla)

        self.ventana1.mainloop()

    def presion\_tecla(self, evento):

        x1, y1, x2, y2 = self.canvas1.coords(self.cuadrado)

        if evento.keysym=='Right':

            if x2+4<=600:

                self.canvas1.move(self.cuadrado, 4, 0)

        if evento.keysym=='Left':

            if x1-4>=0:

                self.canvas1.move(self.cuadrado, -4, 0)

        if evento.keysym=='Down':

            if y2+4<=400:

                self.canvas1.move(self.cuadrado, 0, 4)

        if evento.keysym=='Up':

            if y1-4>=0:

                self.canvas1.move(self.cuadrado, 0, -4)

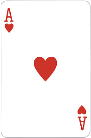
aplicacion1=Aplicacion()

# Canvas : mostrar una imagen

Si necesitamos mostrar una imagen dentro de una componente de tipo Canvas disponemos del método 'PhotoImage' que se encuentra en el módulo tkinter para su lectura, y luego mediante el método 'create\_image' de la clase Canvas para su visualización.

### Problema:

Se cuenta con tres archivos de tipo png con las imágenes de distintas cartas. Mostrarlas a cada una dentro de una componente de tipo Canvas.

Puede descargar estas tres imágenes y copiarlas en la carpeta donde codifica sus programas en Python:  
  

La interfaz visual debe ser similar a esta:

### Canvas PhotoImage create_image Acotaciones

Los formatos reconocidos de la clase PhotoImage son: GIF, PNG, PGM y PPM.

Si el archivo se encuentra en otra carpeta debemos indicar el path del mismo:

#### Programa: ejercicio263.py

import tkinter as tk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.canvas1=tk.Canvas(self.ventana1, width=700, height=500, background="black")

        self.canvas1.grid(column=0, row=0)

        archi1=tk.PhotoImage(file="carta1.png")

        self.canvas1.create\_image(30, 100, image=archi1, anchor="nw")

        archi2=tk.PhotoImage(file="carta2.png")

        self.canvas1.create\_image(240, 100, image=archi2, anchor="nw")

        archi3=tk.PhotoImage(file="carta3.png")

        self.canvas1.create\_image(450, 100, image=archi3, anchor="nw")

        self.ventana1.mainloop()

aplicacion1=Aplicacion()

## Problema propuesto

* Disponer un botón y mostrar al azar una de las tres cartas del problema anterior. Cada vez que se presione el botón generar un valor aleatorio y a partir de dicho valor mostrar una carta.
* ejercicio264.py
* import tkinter as tk
* from tkinter import ttk
* import random
* class Aplicacion:
* def \_\_init\_\_(self):
* self.ventana1=tk.Tk()
* self.boton1=ttk.Button(self.ventana1, text="Sortear", command=self.sortear)
* self.boton1.grid(column=0, row=0)
* self.canvas1=tk.Canvas(self.ventana1, width=300, height=500, background="black")
* self.canvas1.grid(column=0, row=1)
* self.archi1=tk.PhotoImage(file="carta1.png")
* self.archi2=tk.PhotoImage(file="carta2.png")
* self.archi3=tk.PhotoImage(file="carta3.png")
* self.canvas1.create\_image(50, 100, image=self.archi1, anchor="nw")
* self.ventana1.mainloop()
* def sortear(self):
* valor=random.randint(1,3)
* if valor==1:
* self.canvas1.create\_image(50, 100, image=self.archi1, anchor="nw")
* if valor==2:
* self.canvas1.create\_image(50, 100, image=self.archi2, anchor="nw")
* if valor==3:
* self.canvas1.create\_image(50, 100, image=self.archi3, anchor="nw")
* aplicacion1=Aplicacion()

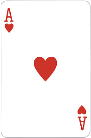
# Canvas : mover una figura

Otro algoritmo útil cuando trabajamos con figuras dentro de un control Canvas es la posibilidad de arrastrar y soltar figuras.

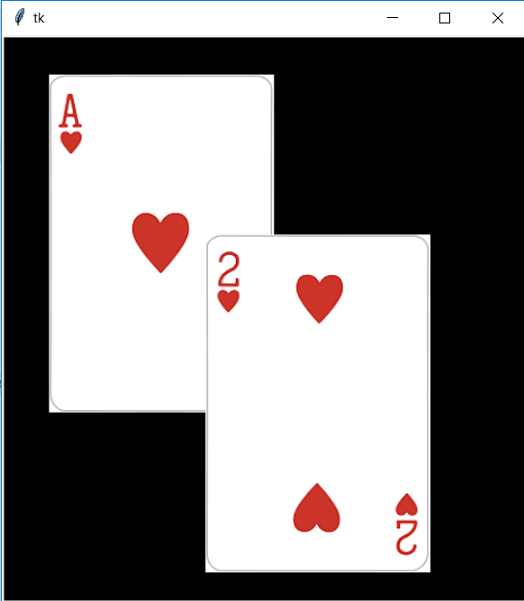
Debemos capturar los eventos cuando se hace clic dentro de una figura y luego cada vez que se desplaza la flecha del mouse.

### Problema:

Se cuenta con dos archivos de tipo png con las imágenes de distintas cartas. Mostrarlas a cada una dentro de una componente de tipo Canvas y permitir moverlas dentro del control mediante el mouse.

Puede descargar estas dos imágenes y copiarlas en la carpeta donde codifica sus programas en Python:  
 

La interfaz visual debe ser similar a esta luego de mover con el mouse las imágenes:



#### Programa: ejercicio265.py

import tkinter as tk

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.canvas1=tk.Canvas(self.ventana1, width=900, height=500, background="black")

        self.canvas1.grid(column=0, row=0)

        archi1=tk.PhotoImage(file="carta1.png")

        self.canvas1.create\_image(30, 100, image=archi1, anchor="nw", tags="movil")

        archi2=tk.PhotoImage(file="carta2.png")

        self.canvas1.create\_image(400, 100, image=archi2, anchor="nw", tags="movil")

        self.canvas1.tag\_bind("movil", "<ButtonPress-1>", self.presion\_boton)

        self.canvas1.tag\_bind("movil", "<Button1-Motion>", self.mover)

        self.carta\_seleccionada = None

        self.ventana1.mainloop()

    def presion\_boton(self, evento):

        carta = self.canvas1.find\_withtag(tk.CURRENT)

        self.carta\_seleccionada = (carta, evento.x, evento.y)

    def mover(self, evento):

        x, y = evento.x, evento.y

        carta, x1, y1 = self.carta\_seleccionada

        self.canvas1.move(carta, x - x1, y - y1)

        self.carta\_seleccionada = (carta, x, y)

aplicacion1=Aplicacion()

## Problema propuesto

* Crear 100 cuadrados de color rojo y disponerlos en el control Canvas en posiciones aleatorias. Permitir desplazar con el mouse cualquiera de los cuadrados.

ejercicio266.py

import tkinter as tk

import random

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.canvas1=tk.Canvas(self.ventana1, width=900, height=500, background="black")

        self.canvas1.grid(column=0, row=0)

        for x in range(101):

            x1=random.randint(1,900)

            y1=random.randint(1,500)

            self.cuadrado=self.canvas1.create\_rectangle(x1, y1, x1+20, y1+20, fill="red", outline="red", tags="movil")

        self.canvas1.tag\_bind("movil", "<ButtonPress-1>", self.presion\_boton)

        self.canvas1.tag\_bind("movil", "<Button1-Motion>", self.mover)

        self.carta\_seleccionada = None

        self.ventana1.mainloop()

    def presion\_boton(self, evento):

        carta = self.canvas1.find\_withtag(tk.CURRENT)

        self.carta\_seleccionada = (carta, evento.x, evento.y)

    def mover(self, evento):

        x, y = evento.x, evento.y

        carta, x1, y1 = self.carta\_seleccionada

        self.canvas1.move(carta, x - x1, y - y1)

        self.carta\_seleccionada = (carta, x, y)

aplicacion1=Aplicacion()

# Archivos de texto: creación, escritura y lectura

Una actividad muy común en un programa es el almacenamiento y recuperación de datos almacenado en un dispositivo secundario (disco duro).

Existen muchos modos de almacenar datos como son los archivos de texto, archivos binarios, bases de datos etc.

Veremos en este concepto como almacenar y recuperar datos de un archivo de texto.

### Archivo de texto

Un archivo de texto contiene un conjunto de caracteres estructurados en distintas líneas. Es un formato de archivo ampliamente utilizado como pueden ser:

* El código fuente de un script en Python se almacena en un archivo de texto (igual que cualquier otro lenguaje de programación)
* Archivos HTML, CSS, XML se almacenan en archivos de texto.
* Archivos JSON etc.

### Creación de un archivo de texto y almacenamiento de datos.

Como es una actividad tan común en todo programa el lenguaje Python incluye por defecto todas las funcionalidades para trabajar con archivos de texto.

Veamos con un ejemplo como crear y almacenar caracteres en un archivo de texto.

### Problema:

Crear un archivo de texto llamado 'datos.txt', almacenar tres líneas de texto. Abrir luego el archivo creado con un editor de texto.

#### Programa: ejercicio267.py

archi1=open("datos.txt","w")

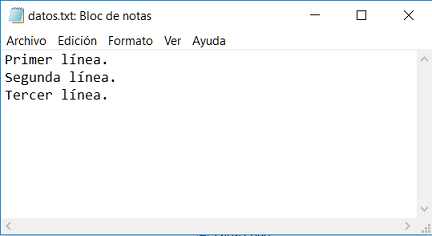
archi1.write("Primer línea.\n")

archi1.write("Segunda línea.\n")

archi1.write("Tercer línea.\n")

archi1.close()

Luego de ejecutar este programa (no muestra nada por pantalla) debemos abrir con un editor de texto el archivo de texto que se acaba de crear llamado 'datos.txt':



Para crear un archivo de texto debemos llamar a la función open y pasar dos parámetros, el primero indica el nombre del archivo a crear y el segundo un string con el caracter "w" (la "w" write indica crear el archivo de texto):

archi1=open("datos.txt","w")

Si el archivo 'datos.txt' ya existe luego se crea uno nuevo y se borra el actual.

El archivo se crea en la misma carpeta donde se está ejecutando el script de Python, si necesitamos que se cree en otra carpeta podemos indicar el path del mismo:

archi1 = open("c:/administracion/datos.txt","w")

Si indicamos un path inexistente se genera un error.

Para grabar caracteres en el archivo de texto utilizamos el método 'write' y le pasamos un string a grabar:

archi1.write("Primer línea.\n")

Mediante la sintaxis \n indicamos que debe almacenarse un salto de línea en el archivo de texto.

Si no incluimos los respectivos \n:

archi1.write("Primer línea.")

archi1.write("Segunda línea.")

archi1.write("Tercer línea.")

### Problema:

Leer el contenido del archivo de texto 'datos.txt'.

#### Programa: ejercicio268.py

archi1=open("datos.txt","r")

contenido=archi1.read()

print(contenido)

archi1.close()

### Lectura de un archivo de texto línea a línea.

En algunas situaciones podemos necesitar leer el contenido de un archivo de texto línea a línea. Disponemos de un método llamado 'readline' que lee una línea completa del archivo, inclusive retorna el caracter '\n' de fin de línea.

### Problema:

Leer el contenido del archivo de texto 'datos.txt' línea a línea.

#### Programa: ejercicio269.py

archi1=open("datos.txt","r")

linea=archi1.readline()

while linea!='':

    print(linea, end='')

    linea=archi1.readline()

archi1.close()

Podemos recorrer el archivo leyendo línea a línea utilizando la estructura repetitiva for:

#### Programa: ejercicio270.py

archi1=open("datos.txt","r")

for linea in archi1:

    print(linea, end='')

archi1.close()

### Almacenar un archivo de texto en una lista

[Ver video](https://youtu.be/8q_kYfHlChg)

Mediante el método 'readlines' podemos recuperar cada una de las líneas del archivo de texto y almacenarlas en una lista.

### Problema:

Leer el contenido del archivo de texto 'datos.txt' y almacenar sus líneas en una lista. Imprimir la cantidad de líneas que tiene el archivo y su contenido.

#### Programa: ejercicio271.py

archi1=open("datos.txt","r")

lineas=archi1.readlines()

print('El archivo tiene', len(lineas), 'líneas')

print('El contenido del archivo')

for linea in lineas:

    print(linea, end='')

archi1.close()

### Abrir un archivo de texto para añadir líneas.

Hemos visto que cuando llamamos a la función 'open' el segundo parámetro puede ser "w", "r" y si queremos que se abra para añadir sin borrar las líneas actuales del archivo debemos hacerlo con el parámetro "a" (append).

### Problema:

Abrir el archivo de texto 'datos.txt' y luego agregar 2 líneas. Imprimir luego el archivo completo.

#### Programa: ejercicio272.py

archi1=open("datos.txt","a")

archi1.write("nueva línea 1\n")

archi1.write("nueva línea 2\n")

archi1.close()

archi1=open("datos.txt","r")

contenido=archi1.read()

print(contenido)

archi1.close()

### Abrir un archivo para leer y agregar datos.

Hay una cuarta forma de abrir un archivo indicando en el segundo parámetro de la función open el string "r+", con dicha opción podemos leer y escribir.

### Problema:

Abrir un archivo de texto con el parámetro "r+", imprimir su contenido actual y agregar luego dos líneas al final.

#### Programa: ejercicio273.py

archi1=open("datos.txt","r+")

contenido=archi1.read()

print(contenido)

archi1.write("Otra línea 1\n")

archi1.write("Otra línea 2\n")

archi1.close()

### Codificación de caracteres utf-8.

Actualmente se utiliza mucho la codificación de caracteres utf-8 la que nos permite representar una infinidad de caracteres de distintos idiomas y símbolos.

En Python debemos indicar cuando abrimos el archivo de texto mediante el parámetro 'encoding' la codificación de caracteres utilizada.

### Problema:

Crear un archivo de texto llamado 'datos.txt' con una codificación utf-8, almacenar tres líneas de texto. Abrir luego el archivo creado con el editor VS Code.

#### Programa: ejercicio274.py

archi1=open("datos.txt","w", encoding="utf-8")

archi1.write("Primer línea.\n")

archi1.write("Segunda línea.\n")

archi1.write("Tercer línea.\n")

archi1.close()

Archivos de texto: interfaz visual con tkinter para leer y escribir un archivo

Conociendo como poder crear, leer y escribir en un archivo de texto, vamos a combinarlo con los conceptos de la librería visual tkinter para implementar un programa que nos permita escribir un texto y guardarlo en un archivo de texto.

Problema:

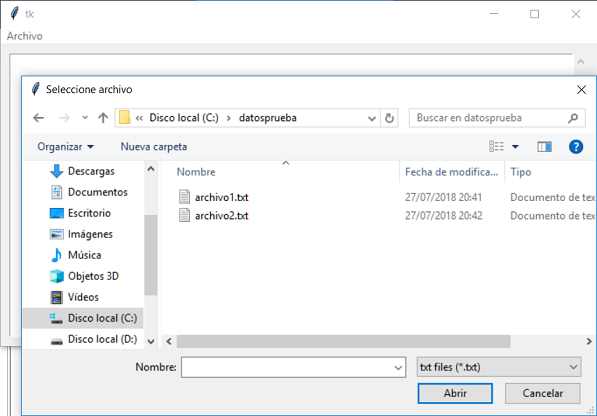
Confeccionar una interfaz visual que contenga un menú de opciones que permitan "Guardar archivo", "Recuperar archivo" y "Salir del programa".

En la ventana principal debe aparecer un control de tipo 'scrolledtext' donde el operador pueda escribir un texto para luego grabarlo en un archivo de texto. También el control 'scrolledtext' debe cargarse con el contenido de un archivo existente en el disco duro.

La interfaz visual debe ser:

Cuando se selecciona la opción "Guardar archivo" debe aparecer el diálogo siguiente:

Y cuando se selecciona la opción "Recuperar archivo" debe aparecer el diálogo:



Programa: ejercicio275.py

import tkinter as tk

from tkinter import scrolledtext as st

import sys

from tkinter import filedialog as fd

from tkinter import messagebox as mb

class Aplicacion:

    def \_\_init\_\_(self):

        self.ventana1=tk.Tk()

        self.agregar\_menu()

        self.scrolledtext1=st.ScrolledText(self.ventana1, width=80, height=20)

        self.scrolledtext1.grid(column=0,row=0, padx=10, pady=10)

        self.ventana1.mainloop()

    def agregar\_menu(self):

        menubar1 = tk.Menu(self.ventana1)

        self.ventana1.config(menu=menubar1)

        opciones1 = tk.Menu(menubar1, tearoff=0)

        opciones1.add\_command(label="Guardar archivo", command=self.guardar)

        opciones1.add\_command(label="Recuperar archivo", command=self.recuperar)

        opciones1.add\_separator()

        opciones1.add\_command(label="Salir", command=self.salir)

        menubar1.add\_cascade(label="Archivo", menu=opciones1)

    def salir(self):

        sys.exit()

    def guardar(self):

        nombrearch=fd.asksaveasfilename(initialdir = "/",title = "Guardar como",filetypes = (("txt files","\*.txt"),("todos los archivos","\*.\*")))

        if nombrearch!='':

            archi1=open(nombrearch, "w", encoding="utf-8")

            archi1.write(self.scrolledtext1.get("1.0", tk.END))

            archi1.close()

            mb.showinfo("Información", "Los datos fueron guardados en el archivo.")

    def recuperar(self):

        nombrearch=fd.askopenfilename(initialdir = "/",title = "Seleccione archivo",filetypes = (("txt files","\*.txt"),("todos los archivos","\*.\*")))

        if nombrearch!='':

            archi1=open(nombrearch, "r", encoding="utf-8")

            contenido=archi1.read()

            archi1.close()

            self.scrolledtext1.delete("1.0", tk.END)

            self.scrolledtext1.insert("1.0", contenido)

aplicacion1=Aplicacion()

MySQL : Base de datos desde Python

Cuando tenemos que almacenar gran cantidad de datos y su posterior procesamiento es muy común utilizar un gestor de bases de datos.  
Con Python podemos comunicarnos con un gestor de bases de datos para enviar y recuperar datos.

Existen gran cantidad de gestores de bases de datos y el primero que veremos para ver cual es la mecánica para conectarnos desde Python será el gestor de base de datos MySQL.

Deberemos instalar primero si no lo tenemos a MySQL.

### Instalación de MySQL.

Para facilitar la administración del MySQL utilizaremos el programa XAMPP que entre otros instala:

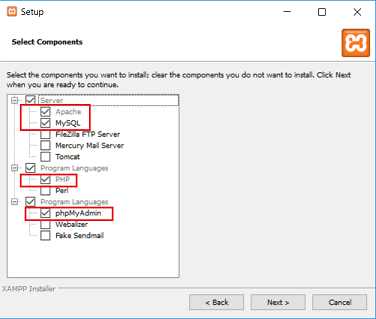
* MySQL
* PHPMyAdmin (que nos permitirá administrar las bases de datos existentes en MySQL)
* PHP (Lenguaje que nos permite ejecutar el PHPMyAdmin)
* Apache (Servidor Web que nos permite ejecutar PHPMyAdmin y PHP en un servidor)

Descargamos e instalamos [XAMPP](https://www.apachefriends.org/es/index.html) para el sistema operativo que estamos trabajando.

Los pasos para instalar son muy sencillos:



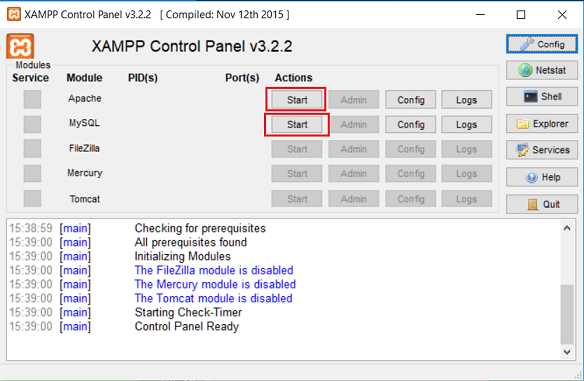
Lo mínimo que debemos instalar es:



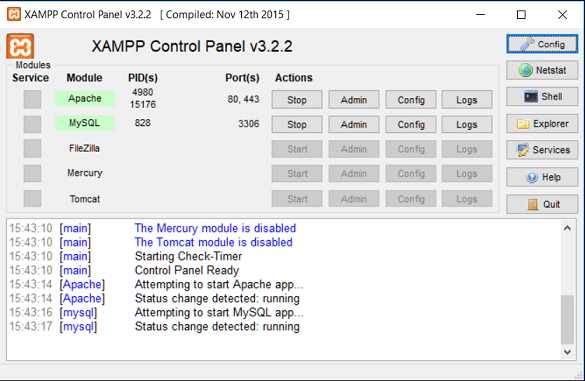
Una vez finalizada la instalación de todo el software debemos arrancar el programa "XAMPP Control Panel":



Aparece la interfaz desde donde debemos iniciar el gestor de base de datos MySQL:



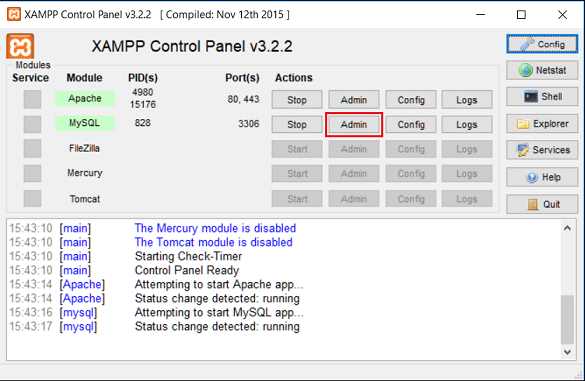
Presionamos el botón "Start" tanto para MySQL como para Apache:



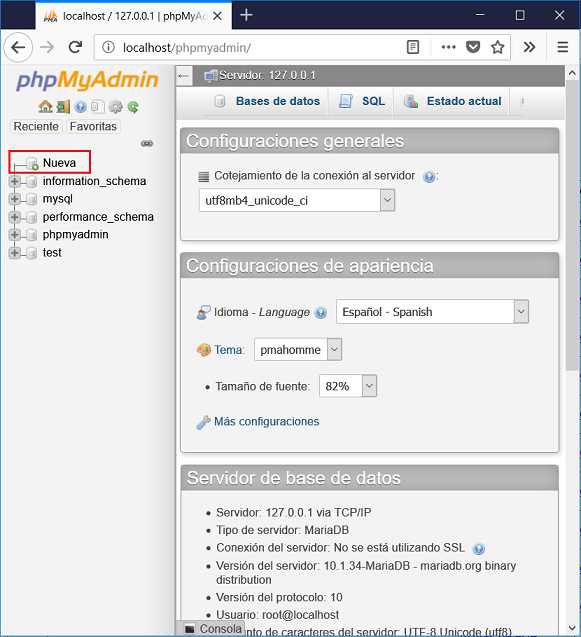
Si aparecen en verde significa que los programas se encuentran correctamente en funcionamiento.

### Creación de la base de datos MySQL.

Para crear la base de datos utilizaremos el programa PHPMyAdmin que lo iniciamos presionando el botón "MySQL - Admin" del "XAMPP Control Panel":

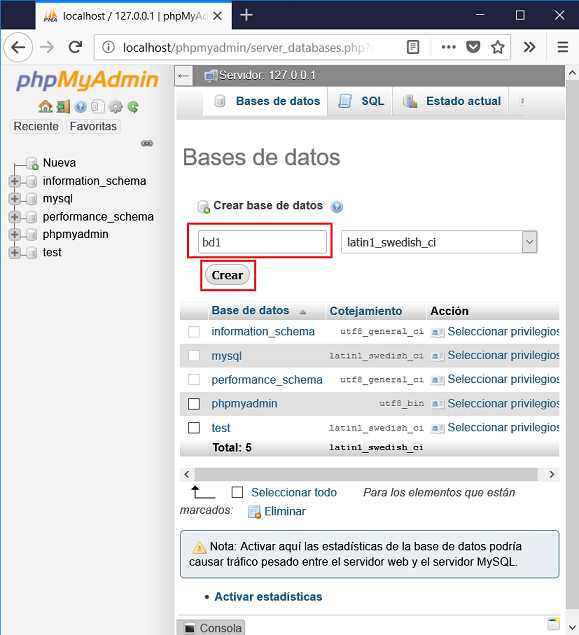


Se nos abre una aplicación web PHPMyAdmin que nos permite administrar nuestras bases de datos de MySQL:

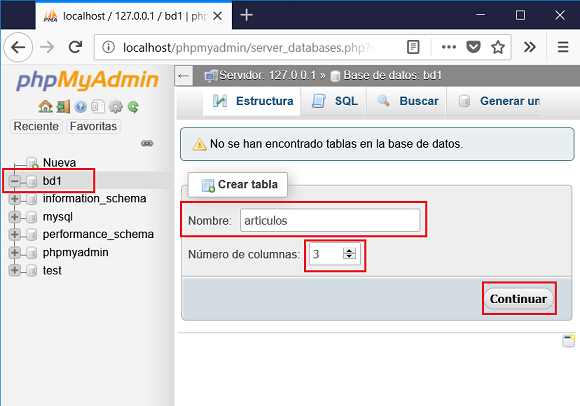


Crearemos una base de datos seleccionando la opción "Nueva" que aparece del lado izquierdo de la página.

En el siguiente diálogo debemos definir el nombre de nuestra base de datos, la llamaremos "bd1":

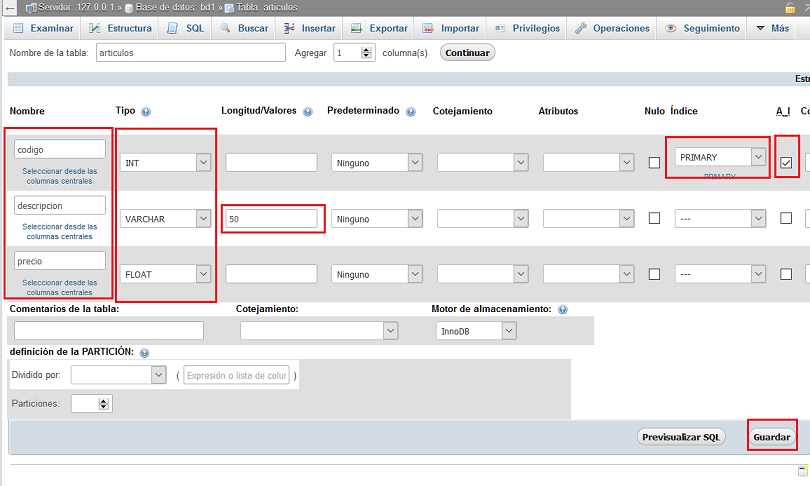


Una vez que se crea la podemos ver que aparece en la columna izquierda y podemos pasar a crear tablas en la misma. Crearemos la tabla artículos que tendrá 3 campos:

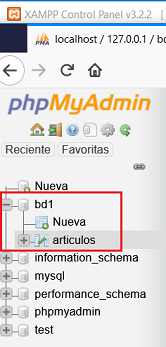


Almacenaremos en la tabla articulos el codigo, descripcion y precio.  
El campo 'codigo' será int'clave primaria' y auto\_increment.  
El campo 'descripcion' será varchar de 50.  
El campo 'precio' será float.

Los datos a ingresar para cada campo para la creación de la tabla articulos son:



Ya tenemos creada la base de datos: "bd1" y en ésta la tabla "articulos":



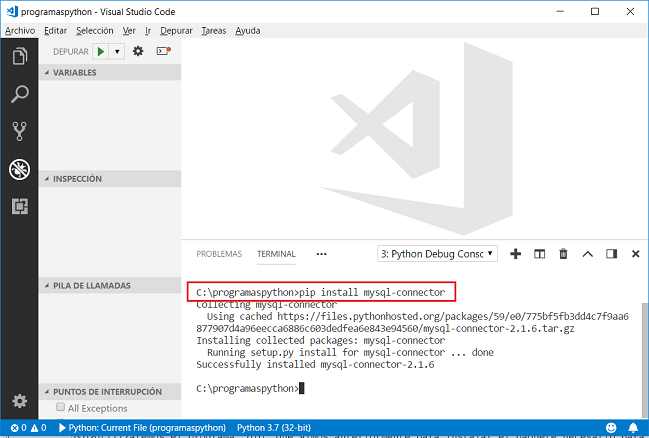
### Paquete de Python necesario para conectarnos a MySQL.

Utilizaremos el programa 'pip' que vimos anteriormente para instalar el paquete necesario para interconectar 'Python' y 'MySQL'.

Desde la línea de comandos ejecutamos el programa pip con el siguiente paquete a instalar:

pip install mysql-connector

Luego de ejecutar el programa pip podemos ver que nos informa de la instalación del paquete 'mysql-connector':



### Conexión con el servidor de MySQL.

Controlar que el "XAMPP Control Panel" se encuentre en ejecución el servidor de MySQL:

El primer programa que implementaremos nos conectaremos con el servidor de MySQL y mostraremos todas las bases de datos existentes (una de esas debería ser bd1)

#### Programa: ejercicio276.py

import pymysql

conexion1=pymysql.connect(host='localhost',

                                user='root',

                                password='',

                                db='bd1')

cursor1=conexion1.cursor()

cursor1.execute("show databases")

for base in cursor1:

    print(base)

conexion1.close()

Listado de todas las tablas de una base de datos de MySQL.

Ahora implementaremos un programa que recupere todas las tablas contenidos en una base de datos. Trabajaremos con la base de datos que creamos desde el PHPMyAdmin llamada 'bd1'.

Programa: ejercicio277.py

import pymysql

conexion1=pymysql.connect(host='localhost',

                                user='root',

                                password='',

                                db='bd1')

cursor1=conexion1.cursor()

cursor1.execute("show tables")

for tabla in cursor1:

    print(tabla)

conexion1.close()

### Insertar filas en una tabla.

Ahora implementaremos un programa que inserte un par de filas en la tabla 'articulos'.

#### Programa: ejercicio278.py

import pymysql

conexion1=pymysql.connect(host='localhost',

                                user='root',

                                password='',

                                db='bd1')

cursor1=conexion1.cursor()

sql="insert into articulos(descripcion, precio) values (%s,%s)"

datos=("naranjas", 23.50)

cursor1.execute(sql, datos)

datos=("peras", 34)

cursor1.execute(sql, datos)

datos=("bananas", 25)

cursor1.execute(sql, datos)

conexion1.commit()

conexion1.close()

Recuperar todas las filas de una tabla.

Implementaremos un programa que solicite ejecutar un 'select' en la tabla 'articulos' y nos retorne todas sus filas.

Programa: ejercicio279.py

import pymysql

conexion1=pymysql.connect(host='localhost',

                                user='root',

                                password='',

                                db='bd1')

cursor1=conexion1.cursor()

cursor1.execute("select codigo, descripcion, precio from articulos")

for fila in cursor1:

    print(fila)

conexion1.close()

### Borrado y modificación de filas.

Las otras dos actividades fundamentales que podemos hacer con una tabla es borrar filas y modificar datos.  
Desarrollaremos un pequeño programa que borre el artículo cuyo código sea el 1 y modifique el precio del artículo cuyo código sea 3.

#### Programa: ejercicio280.py

import pymysql

conexion1=pymysql.connect(host='localhost',

                                user='root',

                                password='',

                                db='bd1')

cursor1=conexion1.cursor()

cursor1.execute("delete from articulos where codigo=1")

cursor1.execute("update articulos set precio=50 where codigo=3")

conexion1.commit()

cursor1.execute("select codigo, descripcion, precio from articulos")

for fila in cursor1:

    print(fila)

conexion1.close()

# MySQL : interfaz visual con tkinter y acceso a la base de datos

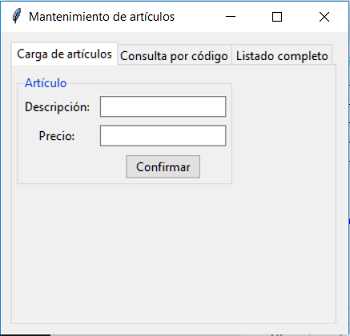
Es muy posible que necesitemos en algunas situaciones acceder a una base de datos de MySQL desde una aplicación con una interfaz visual.

### Problema:

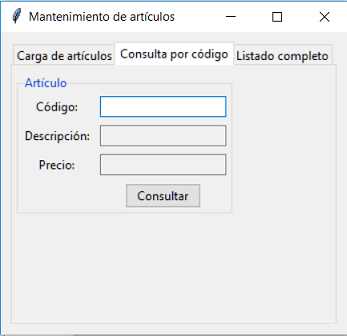
Desarrollar una aplicación visual con la librería tkinter que permita implementar los algoritmos de carga de artículos, consulta por código y listado completo.

Seguiremos trabajando con la tabla 'articulos' que creamos en el concepto anterior.

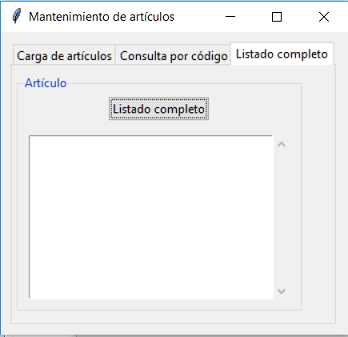
Las interfaz visual para la carga debe ser:



Las interfaz visual para la consulta:



Y finalmente la interfaz para el listado completo:



Para trabajar un poco más ordenado en la resolución de este problema lo dividiremos en dos módulos 'formularioarticulos.py' y 'articulos.py'.

El primer paso será crear una carpeta llamada 'proyecto3' y dentro de esta crearemos los dos módulos:

El módulo 'formularioarticulos.py' contiene toda la lógica de presentación de datos y hace uso del otro módulo para el acceso a la base de datos MySQL.

módulo: formularioarticulos.py

import tkinter as tk

from tkinter import ttk

from tkinter import messagebox as mb

from tkinter import scrolledtext as st

import articulos

class FormularioArticulos:

    def \_\_init\_\_(self):

        self.articulo1=articulos.Articulos()

        self.ventana1=tk.Tk()

        self.ventana1.title("Mantenimiento de artículos")

        self.cuaderno1 = ttk.Notebook(self.ventana1)

        self.carga\_articulos()

        self.consulta\_por\_codigo()

        self.listado\_completo()

        self.cuaderno1.grid(column=0, row=0, padx=10, pady=10)

        self.ventana1.mainloop()

    def carga\_articulos(self):

        self.pagina1 = ttk.Frame(self.cuaderno1)

        self.cuaderno1.add(self.pagina1, text="Carga de artículos")

        self.labelframe1=ttk.LabelFrame(self.pagina1, text="Artículo")

        self.labelframe1.grid(column=0, row=0, padx=5, pady=10)

        self.label1=ttk.Label(self.labelframe1, text="Descripción:")

        self.label1.grid(column=0, row=0, padx=4, pady=4)

        self.descripcioncarga=tk.StringVar()

        self.entrydescripcion=ttk.Entry(self.labelframe1, textvariable=self.descripcioncarga)

        self.entrydescripcion.grid(column=1, row=0, padx=4, pady=4)

        self.label2=ttk.Label(self.labelframe1, text="Precio:")

        self.label2.grid(column=0, row=1, padx=4, pady=4)

        self.preciocarga=tk.StringVar()

        self.entryprecio=ttk.Entry(self.labelframe1, textvariable=self.preciocarga)

        self.entryprecio.grid(column=1, row=1, padx=4, pady=4)

        self.boton1=ttk.Button(self.labelframe1, text="Confirmar", command=self.agregar)

        self.boton1.grid(column=1, row=2, padx=4, pady=4)

    def agregar(self):

        datos=(self.descripcioncarga.get(), self.preciocarga.get())

        self.articulo1.alta(datos)

        mb.showinfo("Información", "Los datos fueron cargados")

        self.descripcioncarga.set("")

        self.preciocarga.set("")

    def consulta\_por\_codigo(self):

        self.pagina2 = ttk.Frame(self.cuaderno1)

        self.cuaderno1.add(self.pagina2, text="Consulta por código")

        self.labelframe2=ttk.LabelFrame(self.pagina2, text="Artículo")

        self.labelframe2.grid(column=0, row=0, padx=5, pady=10)

        self.label1=ttk.Label(self.labelframe2, text="Código:")

        self.label1.grid(column=0, row=0, padx=4, pady=4)

        self.codigo=tk.StringVar()

        self.entrycodigo=ttk.Entry(self.labelframe2, textvariable=self.codigo)

        self.entrycodigo.grid(column=1, row=0, padx=4, pady=4)

        self.label2=ttk.Label(self.labelframe2, text="Descripción:")

        self.label2.grid(column=0, row=1, padx=4, pady=4)

        self.descripcion=tk.StringVar()

        self.entrydescripcion=ttk.Entry(self.labelframe2, textvariable=self.descripcion, state="readonly")

        self.entrydescripcion.grid(column=1, row=1, padx=4, pady=4)

        self.label3=ttk.Label(self.labelframe2, text="Precio:")

        self.label3.grid(column=0, row=2, padx=4, pady=4)

        self.precio=tk.StringVar()

        self.entryprecio=ttk.Entry(self.labelframe2, textvariable=self.precio, state="readonly")

        self.entryprecio.grid(column=1, row=2, padx=4, pady=4)

        self.boton1=ttk.Button(self.labelframe2, text="Consultar", command=self.consultar)

        self.boton1.grid(column=1, row=3, padx=4, pady=4)

    def consultar(self):

        datos=(self.codigo.get(), )

        respuesta=self.articulo1.consulta(datos)

        if len(respuesta)>0:

            self.descripcion.set(respuesta[0][0])

            self.precio.set(respuesta[0][1])

        else:

            self.descripcion.set('')

            self.precio.set('')

            mb.showinfo("Información", "No existe un artículo con dicho código")

    def listado\_completo(self):

        self.pagina3 = ttk.Frame(self.cuaderno1)

        self.cuaderno1.add(self.pagina3, text="Listado completo")

        self.labelframe3=ttk.LabelFrame(self.pagina3, text="Artículo")

        self.labelframe3.grid(column=0, row=0, padx=5, pady=10)

        self.boton1=ttk.Button(self.labelframe3, text="Listado completo", command=self.listar)

        self.boton1.grid(column=0, row=0, padx=4, pady=4)

        self.scrolledtext1=st.ScrolledText(self.labelframe3, width=30, height=10)

        self.scrolledtext1.grid(column=0,row=1, padx=10, pady=10)

    def listar(self):

        respuesta=self.articulo1.recuperar\_todos()

        self.scrolledtext1.delete("1.0", tk.END)

        for fila in respuesta:

            self.scrolledtext1.insert(tk.END, "código:"+str(fila[0])+"\ndescripción:"+fila[1]+"\nprecio:"+str(fila[2])+"\n\n")

aplicacion1=FormularioArticulos()

El módulo 'articulos.py' contiene toda la lógica de acceso a datos.

módulo: articulos.py

import pymysql

class Articulos:

    def abrir(self):

        conexion=pymysql.connect(host="localhost",

                                              user="root",

                                              passwd="",

                                              database="bd1")

        return conexion

    def alta(self, datos):

        cone=self.abrir()

        cursor=cone.cursor()

        sql="insert into articulos(descripcion, precio) values (%s,%s)"

        cursor.execute(sql, datos)

        cone.commit()

        cone.close()

    def consulta(self, datos):

        cone=self.abrir()

        cursor=cone.cursor()

        sql="select descripcion, precio from articulos where codigo=%s"

        cursor.execute(sql, datos)

        cone.close()

        return cursor.fetchall()

    def recuperar\_todos(self):

        cone=self.abrir()

        cursor=cone.cursor()

        sql="select codigo, descripcion, precio from articulos"

        cursor.execute(sql)

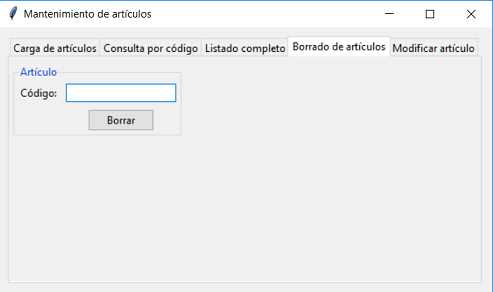
        cone.close()

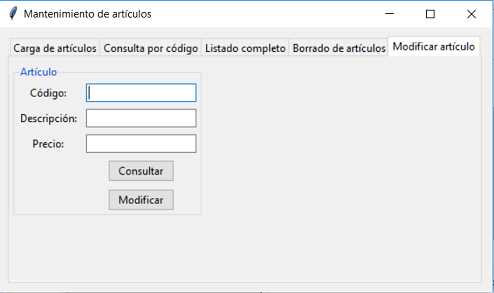
        return cursor.fetchall()

## Problema propuesto

* Agregar dos pestañas al programa de administración de artículos que permitan borrar un artículo ingresando su código y otra opción que permita consultar y modificar la descripción y precio de un artículo.

La interfaces visuales a implementar son:





**Eso se los dejo de tarea**

# otras funcionalidades del paquete mysql-connector

Hemos resuelto en los dos conceptos anteriores algoritmos en Python que acceden a una base de datos de MySQL. Veremos otras funcionalidades que nos provee el paquete mysql-connector.

Recuperar el valor del campo auto\_increment en un insert.

Cuando insertamos una fila en una tabla que contiene un campo que se auto incrementa podemos recuperar dicho valor en el mismo momento que efectuamos la inserción.

Programa: ejercicio281.py

from multiprocessing.connection import Connection

import pymysql

conexion1=pymysql.connect(host="localhost",

                                  user="root",

                                  passwd="",

                                  database="bd1")

cursor1=conexion1.cursor()

sql="insert into articulos(descripcion, precio) values (%s,%s)"

datos=("naranjas", 23.50)

cursor1.execute(sql, datos)

conexion1.commit()

print("Valor del último codigo de artículo generado:", cursor1.lastrowid)

conexion1.close()

Inserción de múltiples filas en una tabla.

La clase cursor a parte del método 'execute' cuenta con otro método llamado 'executemany' que tiene el objetivo de insertar múltiples filas de una tabla.

Programa: ejercicio282.py

import pymysql

conexion1=pymysql.connect(host="localhost",

                                  user="root",

                                  passwd="",

                                  database="bd1")

cursor1=conexion1.cursor()

sql="insert into articulos(descripcion, precio) values (%s,%s)"

filas=[ ("naranjas", 23.50),

        ("bananas", 34),

        ("peras", 21),

        ("sandía", 19.60) ]

cursor1.executemany(sql, filas)

conexion1.commit()

conexion1.close()

Creación de una base de datos y tablas.

En conceptos anteriores vimos como crear una base de datos de MySQL utilizando la aplicación PHPMyAdmin, en algunas situaciones podemos necesitar crear una base de datos desde el mismo programa de Python. La misma metodología será si queremos crear tablas.

Programa: ejercicio283.py

import pymysql

conexion1=pymysql.connect(host="localhost",

                                  user="root",

                                  passwd="")

cursor1=conexion1.cursor()

sql="create database bd2"

cursor1.execute(sql)

sql="use bd2"

cursor1.execute(sql)

sql="""create table usuarios (

         nombre varchar(30) primary key,

         clave  varchar(30)

   )"""

cursor1.execute(sql)

conexion1.commit()

conexion1.close()

Se ejecutamos nuevamente el programa veremos que se genera un error indicando que la base de datoas 'bd2' ya existe.

Borrar una base de datos.

Cuando queremos crear una base de datos que ya existe se genera un error, podemos primero borrarla y luego ya si crearla sin problemas.

Programa: ejercicio284.py

import pymysql

conexion1=pymysql.connect(host="localhost",

                                  user="root",

                                  passwd="")

cursor1=conexion1.cursor()

sql="drop database if exists bd2"

cursor1.execute(sql)

sql="create database bd2"

cursor1.execute(sql)

sql="use bd2"

cursor1.execute(sql)

sql="""create table usuarios (

         nombre varchar(30) primary key,

         clave  varchar(30)

   )"""

cursor1.execute(sql)

conexion1.commit()

conexion1.close()

### Borrar una tabla y crear otra con el mismo nombre.

No se puede crear una tabla con el mismo nombre de una existente, en esos casos debemos borrar la actual y crear una nueva.

#### Programa: ejercicio285.py

import pymysql

conexion1=pymysql.connect(host="localhost",

                                  user="root",

                                  passwd="",

                                  database="bd2")

cursor1=conexion1.cursor()

sql="drop table if exists usuarios"

cursor1.execute(sql)

sql="""create table usuarios (

         nombre varchar(30) primary key,

         clave  varchar(30)

   )"""

cursor1.execute(sql)

conexion1.commit()

conexion1.close()

# SQLite : Base de datos desde Python

[Ver video](https://youtu.be/eQjL2AdoN5k)

La segunda base de datos que veremos su acceso desde Python es SQLite.

Se encuentra invitado para desarrollar un curso completo de [SQLite Ya!](https://www.tutorialesprogramacionya.com/sqliteya/)

SQLite también es un gestor de bases de datos relacional pero con objetivos muy diferentes a MySQL, SQLServer, Oracle etc.  
Este gestor de base de datos tiene por objetivo ser parte de la misma aplicación con la que colabora, es decir no cumple los conceptos de cliente y servidor.

Para entender sus usos podemos dar algunos ejemplos donde se utiliza el gestor SQLite:

* Firefox usa SQLite para almacenar los favoritos, el historial, las cookies etc.
* También el navegador Opera usa SQLite.
* La aplicación de comunicaciones Skype de Microsoft utiliza SQLite
* Los sistemas operativos Android y iOS adoptan SQLite para permitir el almacenamiento y recuperación de datos.

SQLite es Open Source y se ha instalado por defecto con Python, es decir forma parte de la biblioteca estándar, no tenemos que instalar ningún módulo con pip.

Si nuestra aplicación necesita almacenar gran cantidad de información local con cierta estructura el empleo de SQLite es nuestra principal opción.

### Creación de una base de datos y tablas.

En principio no se requiere tener más que Python instalado para poder trabajar con SQLite. Podemos desde nuestra propia aplicación crear la base de datos y sus tablas.

#### Programa: ejercicio303.py

import sqlite3

conexion=sqlite3.connect("bd1.db")

try:

conexion.execute("""create table articulos (

codigo integer primary key autoincrement,

descripcion text,

precio real

)""")

print("se creo la tabla articulos")

except sqlite3.OperationalError:

print("La tabla articulos ya existe")

conexion.close()

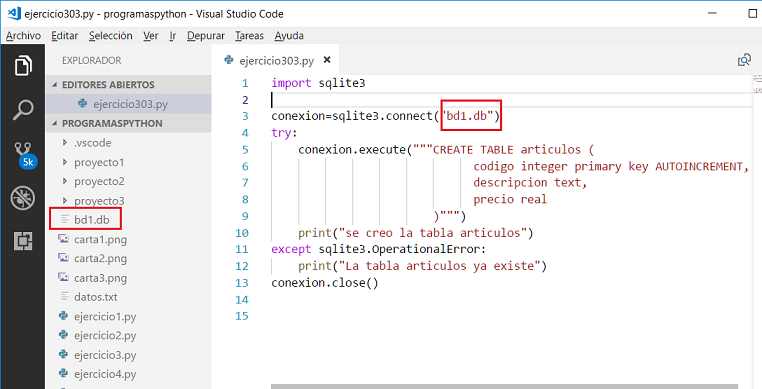
Para poder trabajar con bases de datos de tipo SQLite debemos primero importar el módulo 'sqlite3':

import sqlite3

Para crear o abrir una conexión con una base de datos existente debemos llamar a la función 'connect' del módulo 'sqlite3':

conexion=sqlite3.connect("bd1.db")

La primera vez que ejecutemos este programa como no existe la base de datos 'bd1.db' se crea, consiste en un único archivo que se localiza en la misma carpeta de nuestra aplicación:



Disponemos un try/except al momento de crear la tabla debido a que si ejecutamos por segunda vez este programa se tratará de crear nuevamente la tabla 'articulos' y al ya existir se genera una excepción de tipo 'OperationalError':

try:

conexion.execute("""create table articulos (

codigo integer primary key autoincrement,

descripcion text,

precio real

)""")

print("se creo la tabla articulos")

except sqlite3.OperationalError:

print("La tabla articulos ya existe")

Si no queremos disponer la excepción 'OperationalError' podemos modificar el comando SQL de la creación de la tabla con la sintaxis:

import sqlite3

conexion=sqlite3.connect("bd1.db")

conexion.execute("""create table if not exists articulos (

codigo integer primary key AUTOINCREMENT,

descripcion text,

precio real

)""")

conexion.close()

### Insertar filas en una tabla.

Ahora implementaremos un programa que inserte un par de filas en la tabla 'articulos' de la base de datos 'bd1' que acabamos de crear con el programa anterior.

#### Programa: ejercicio304.py

import sqlite3

conexion=sqlite3.connect("bd1.db")

conexion.execute("insert into articulos(descripcion,precio) values (?,?)", ("naranjas", 23.50))

conexion.execute("insert into articulos(descripcion,precio) values (?,?)", ("peras", 34))

conexion.execute("insert into articulos(descripcion,precio) values (?,?)", ("bananas", 25))

conexion.commit()

conexion.close()

Llamamos a execute y le pasamos como primer parámetro un comando SQL 'insert' con el caracter '?' indicamos las posiciones donde se van a sustituir. El segundo parámetro es una tupla con los datos que se utilizarán en la sustitución:

conexion.execute("insert into articulos(descripcion,precio) values (?,?)", ("naranjas", 23.50))

Luego de efectuar todos los insert debemos llamar a 'commit' para que se actualicen los datos realmente en la tabla de la base de datos:

conexion.commit()

### Recuperar todas las filas de una tabla.

Implementaremos un programa que solicite ejecutar un 'select' en la tabla 'articulos' y nos retorne todas sus filas.

#### Programa: ejercicio305.py

import sqlite3

conexion=sqlite3.connect("bd1.db")

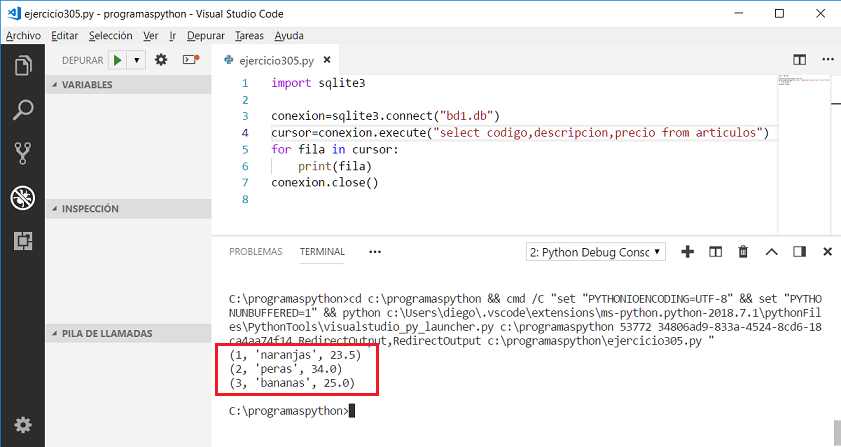
cursor=conexion.execute("select codigo,descripcion,precio from articulos")

for fila in cursor:

print(fila)

conexion.close()

Si ejecutamos este programa luego de haber cargado las tres filas del ejercicio anterior el resultado será el siguiente:



El método execute retorna un objeto de la clase Cursor:

cursor=conexion.execute("select codigo,descripcion,precio from articulos")

### Recuperar una fila de una tabla.

Implementaremos un programa que solicite el ingreso del código de un producto y luego nos muestre su descripción y precio.

#### Programa: ejercicio306.py

import sqlite3

conexion=sqlite3.connect("bd1.db")

codigo=int(input("Ingrese el código de un artículo:"))

cursor=conexion.execute("select descripcion,precio from articulos where codigo=?", (codigo, ))

fila=cursor.fetchone()

if fila!=None:

print(fila)

else:

print("No existe un artículo con dicho código.")

conexion.close()

El resultado de este comando SQL select puede ser de una fila si existe el código de artículo ingresado o cero filas:

cursor=conexion.execute("select codigo,descripcion,precio from articulos where codigo=?", (codigo, ))

fila=cursor.fetchone()

El método fechone de la clase Cursor retorna una tupla con la fila de la tabla que coincide con el código ingresado o retorna 'None':

fila=cursor.fetchone()

if fila!=None:

print(fila)

else:

print("No existe un artículo con dicho código.")

### Recuperar varias filas de una tabla.

Implementaremos un programa que solicite el ingreso de un precio y luego nos muestre la descripción de todos los artículos con un precio inferior al ingresado.

#### Programa: ejercicio307.py

import sqlite3

conexion=sqlite3.connect("bd1.db")

precio=float(input("Ingrese un precio:"))

cursor=conexion.execute("select descripcion from articulos where precio<?", (precio, ))

filas=cursor.fetchall()

if len(filas)>0:

for fila in filas:

print(fila)

else:

print("No existen artículos con un precio menor al ingresado.")

conexion.close()

En este caso el resultado del comando 'select' pueden ser muchas filas:

cursor=conexion.execute("select descripcion from articulos where precio<?", (precio, ))

Llamamos al método 'fetchall' de la clase Cursor y nos retorna una lista con todas las filas de la tabla que cumplen la condición de tener un precio inferior al ingresado:

filas=cursor.fetchall()

En el caso que la lista no esté vacía procedemos a imprimirla:

if len(filas)>0:

for fila in filas:

print(fila)

else:

print("No existen artículos con un precio menor al ingresado.")

[Retornar](https://www.tutorialesprogramacionya.com/pythonya/index.php?inicio=75)

# SQLite : interfaz visual con tkinter y acceso a la base de datos

Es muy posible que necesitemos en algunas situaciones acceder a una base de datos de SQLite desde una aplicación con una interfaz visual.  
Implementaremos el mismo problema que resolvimos cuando trabajamos con el gestor de base de datos MySQL.

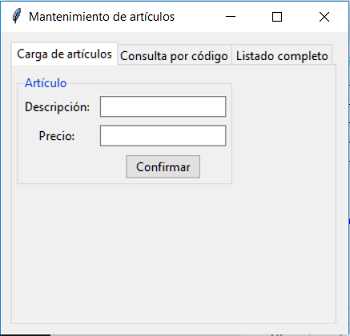
### Problema:

[Ver video](https://youtu.be/vcyvDyQ8QXQ)

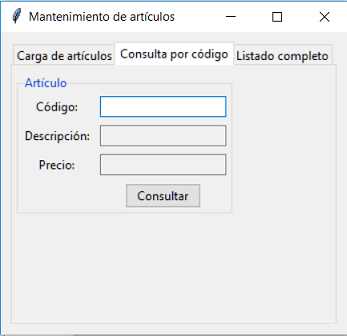
Desarrollar una aplicación visual con la librería tkinter que permita implementar los algoritmos de carga de artículos, consulta por código y listado completo.

Trabajaremos con la base de datos 'bd1.db' que creamos en el concepto anterior.

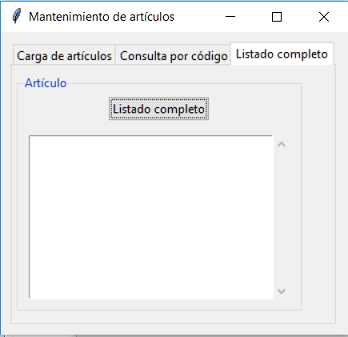
Las interfaz visual para la carga debe ser:



Las interfaz visual para la consulta:

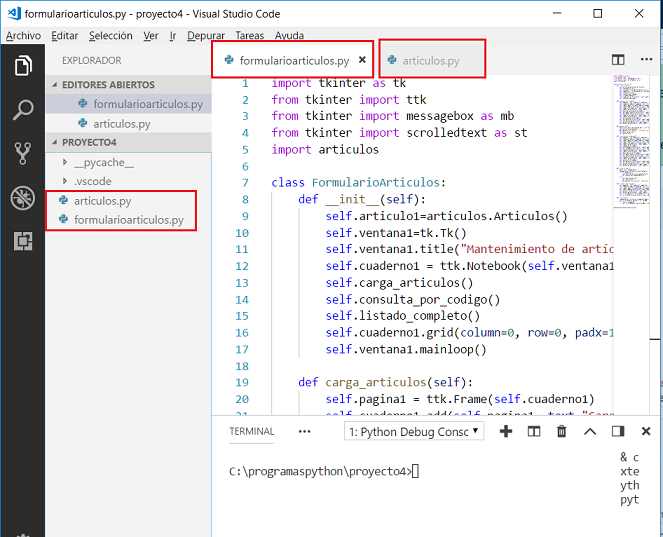


Y finalmente la interfaz para el listado completo:



Para trabajar un poco más ordenado en la resolución de este problema lo dividiremos en dos módulos 'formularioarticulos.py' y 'articulos.py'.

El primer paso será crear una carpeta llamada 'proyecto4' y dentro de esta crearemos los dos módulos:



El módulo 'formularioarticulos.py' contiene toda la lógica de presentación de datos y hace uso del otro módulo para el acceso a la base de datos SQLite.

#### módulo: formularioarticulos.py

import tkinter as tk

from tkinter import ttk

from tkinter import messagebox as mb

from tkinter import scrolledtext as st

import articulos

class FormularioArticulos:

def \_\_init\_\_(self):

self.articulo1=articulos.Articulos()

self.ventana1=tk.Tk()

self.ventana1.title("Mantenimiento de artículos")

self.cuaderno1 = ttk.Notebook(self.ventana1)

self.carga\_articulos()

self.consulta\_por\_codigo()

self.listado\_completo()

self.cuaderno1.grid(column=0, row=0, padx=10, pady=10)

self.ventana1.mainloop()

def carga\_articulos(self):

self.pagina1 = ttk.Frame(self.cuaderno1)

self.cuaderno1.add(self.pagina1, text="Carga de artículos")

self.labelframe1=ttk.LabelFrame(self.pagina1, text="Artículo")

self.labelframe1.grid(column=0, row=0, padx=5, pady=10)

self.label1=ttk.Label(self.labelframe1, text="Descripción:")

self.label1.grid(column=0, row=0, padx=4, pady=4)

self.descripcioncarga=tk.StringVar()

self.entrydescripcion=ttk.Entry(self.labelframe1, textvariable=self.descripcioncarga)

self.entrydescripcion.grid(column=1, row=0, padx=4, pady=4)

self.label2=ttk.Label(self.labelframe1, text="Precio:")

self.label2.grid(column=0, row=1, padx=4, pady=4)

self.preciocarga=tk.StringVar()

self.entryprecio=ttk.Entry(self.labelframe1, textvariable=self.preciocarga)

self.entryprecio.grid(column=1, row=1, padx=4, pady=4)

self.boton1=ttk.Button(self.labelframe1, text="Confirmar", command=self.agregar)

self.boton1.grid(column=1, row=2, padx=4, pady=4)

def agregar(self):

datos=(self.descripcioncarga.get(), self.preciocarga.get())

self.articulo1.alta(datos)

mb.showinfo("Información", "Los datos fueron cargados")

self.descripcioncarga.set("")

self.preciocarga.set("")

def consulta\_por\_codigo(self):

self.pagina2 = ttk.Frame(self.cuaderno1)

self.cuaderno1.add(self.pagina2, text="Consulta por código")

self.labelframe2=ttk.LabelFrame(self.pagina2, text="Artículo")

self.labelframe2.grid(column=0, row=0, padx=5, pady=10)

self.label1=ttk.Label(self.labelframe2, text="Código:")

self.label1.grid(column=0, row=0, padx=4, pady=4)

self.codigo=tk.StringVar()

self.entrycodigo=ttk.Entry(self.labelframe2, textvariable=self.codigo)

self.entrycodigo.grid(column=1, row=0, padx=4, pady=4)

self.label2=ttk.Label(self.labelframe2, text="Descripción:")

self.label2.grid(column=0, row=1, padx=4, pady=4)

self.descripcion=tk.StringVar()

self.entrydescripcion=ttk.Entry(self.labelframe2, textvariable=self.descripcion, state="readonly")

self.entrydescripcion.grid(column=1, row=1, padx=4, pady=4)

self.label3=ttk.Label(self.labelframe2, text="Precio:")

self.label3.grid(column=0, row=2, padx=4, pady=4)

self.precio=tk.StringVar()

self.entryprecio=ttk.Entry(self.labelframe2, textvariable=self.precio, state="readonly")

self.entryprecio.grid(column=1, row=2, padx=4, pady=4)

self.boton1=ttk.Button(self.labelframe2, text="Consultar", command=self.consultar)

self.boton1.grid(column=1, row=3, padx=4, pady=4)

def consultar(self):

datos=(self.codigo.get(), )

respuesta=self.articulo1.consulta(datos)

if len(respuesta)>0:

self.descripcion.set(respuesta[0][0])

self.precio.set(respuesta[0][1])

else:

self.descripcion.set('')

self.precio.set('')

mb.showinfo("Información", "No existe un artículo con dicho código")

def listado\_completo(self):

self.pagina3 = ttk.Frame(self.cuaderno1)

self.cuaderno1.add(self.pagina3, text="Listado completo")

self.labelframe3=ttk.LabelFrame(self.pagina3, text="Artículo")

self.labelframe3.grid(column=0, row=0, padx=5, pady=10)

self.boton1=ttk.Button(self.labelframe3, text="Listado completo", command=self.listar)

self.boton1.grid(column=0, row=0, padx=4, pady=4)

self.scrolledtext1=st.ScrolledText(self.labelframe3, width=30, height=10)

self.scrolledtext1.grid(column=0,row=1, padx=10, pady=10)

def listar(self):

respuesta=self.articulo1.recuperar\_todos()

self.scrolledtext1.delete("1.0", tk.END)

for fila in respuesta:

self.scrolledtext1.insert(tk.END, "código:"+str(fila[0])+"\ndescripción:"+fila[1]+"\nprecio:"+str(fila[2])+"\n\n")

aplicacion1=FormularioArticulos()

El módulo 'articulos.py' contiene toda la lógica de acceso a SQLite.

#### módulo: articulos.py

import sqlite3

class Articulos:

def abrir(self):

conexion=sqlite3.connect("c:/programaspython/bd1.db")

return conexion

def alta(self, datos):

cone=self.abrir()

cursor=cone.cursor()

sql="insert into articulos(descripcion, precio) values (?,?)"

cursor.execute(sql, datos)

cone.commit()

cone.close()

def consulta(self, datos):

try:

cone=self.abrir()

cursor=cone.cursor()

sql="select descripcion, precio from articulos where codigo=?"

cursor.execute(sql, datos)

return cursor.fetchall()

finally:

cone.close()

def recuperar\_todos(self):

try:

cone=self.abrir()

cursor=cone.cursor()

sql="select codigo, descripcion, precio from articulos"

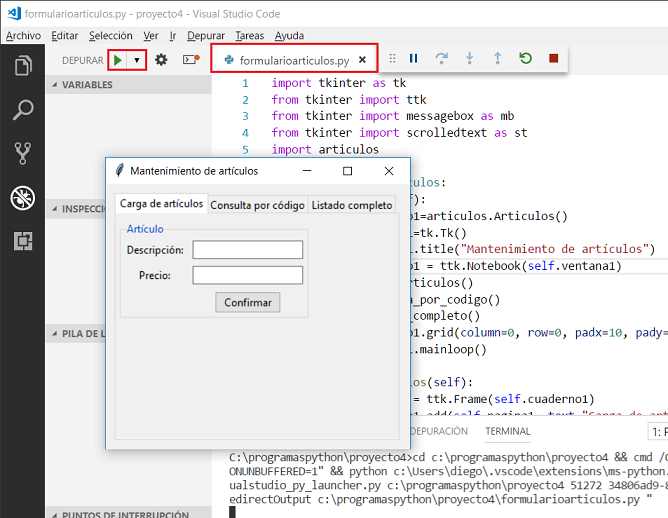
cursor.execute(sql)

return cursor.fetchall()

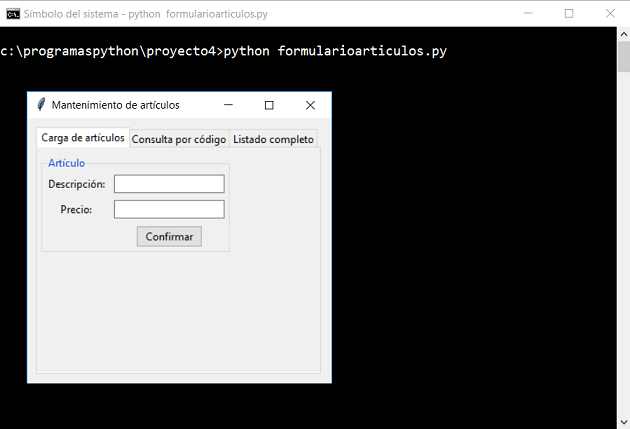
finally:

cone.close()

Tener en cuenta que el módulo principal se encuentra en el archivo 'formularioarticulos.py' y es el que debemos ejecutar:



Cuando al programa lo ejecutemos desde la línea de comandos fuera del editor VS Code debemos recordar de llamar al módulo principal:



Analicemos un poco el código del módulo 'formularioarticulos.py', lo primero que hacemos es importar los módulos necesarios para implementar la interfaz visual:

import tkinter as tk

from tkinter import ttk

from tkinter import messagebox as mb

from tkinter import scrolledtext as st

Otro import fundamental es el módulo 'articulos.py' donde tenemos implementada la clase 'Articulos' que es la que se comunica con la base de datos SQLite:

import articulos

La clase visual la hemos llamado 'FormularioArticulos' y en el método \_\_init\_\_ creamos un objeto de la clase 'Articulos' que se encuentra en el otro módulo:

class FormularioArticulos:

def \_\_init\_\_(self):

self.articulo1=articulos.Articulos()

También en el método \_\_init\_\_ llamamos a una serie de métodos para crear cada una de las páginas del objeto de la clase 'Notebook':

self.ventana1=tk.Tk()

self.ventana1.title("Mantenimiento de artículos")

self.cuaderno1 = ttk.Notebook(self.ventana1)

self.carga\_articulos()

self.consulta\_por\_codigo()

self.listado\_completo()

self.cuaderno1.grid(column=0, row=0, padx=10, pady=10)

self.ventana1.mainloop()

Cuando desde la pestaña "Carga de artículos" se presiona el botón "Confirmar" lo primero que hacemos es crear una tupla con los dos datos ingresados en los controles "Entry":

def agregar(self):

datos=(self.descripcioncarga.get(), self.preciocarga.get())

Luego llamamos al método alta del objeto 'articulo1' y le pasamos la tupla con los datos a añadir:

self.articulo1.alta(datos)

mb.showinfo("Información", "Los datos fueron cargados")

self.descripcioncarga.set("")

self.preciocarga.set("")

En este momento nos conviene analizar el método 'alta' de la clase 'Articulos' que se encuentra en el otro módulo:

def alta(self, datos):

cone=self.abrir()

cursor=cone.cursor()

sql="insert into articulos(descripcion, precio) values (?,?)"

cursor.execute(sql, datos)

cone.commit()

cone.close()

En el método 'alta' abrimos la conexión con el SQLite, creamos un cursor y llamamos seguidamente al método execute pasando un string con el comando SQL 'insert' y los datos a almacenar. Llamamos al método commit de la conexión y finalmente cerramos la conexión.

El método para abrir la conexión es:

def abrir(self):

conexion=sqlite3.connect("c:/programaspython/bd1.db")

return conexion

Es importante especificar el path donde se encuentra el archivo 'bd1.db' si nuestro programa en Python se encuentra en otra carpeta.

#### Consulta por código

Cuando se presiona el botón "Consultar" se ejecuta el método siguiente:

def consultar(self):

datos=(self.codigo.get(), )

respuesta=self.articulo1.consulta(datos)

if len(respuesta)>0:

self.descripcion.set(respuesta[0][0])

self.precio.set(respuesta[0][1])

else:

self.descripcion.set('')

self.precio.set('')

mb.showinfo("Información", "No existe un artículo con dicho código")

Creamos una tubla con un solo dato (es obligatoria la coma para que Python lo considere una tupla:

datos=(self.codigo.get(), )

Llamamos al método consulta de la clase 'Articulos' que se encuentra en el otro módulo. El método 'consulta' retorna una lista vacía si no existe el código de artículo ingresado o una lista con una tupla en su interior.

El método 'consulta' de la clase 'Articulos' llama al método 'fetchall' del cursor respectivo:

def consulta(self, datos):

try:

cone=self.abrir()

cursor=cone.cursor()

sql="select descripcion, precio from articulos where codigo=?"

cursor.execute(sql, datos)

return cursor.fetchall()

finally:

cone.close()

#### Listado completo

Para mostrar todas las filas de la tabla 'articulos' hemos dispuesto un objeto de la clase 'scrolledtext':

def listar(self):

respuesta=self.articulo1.recuperar\_todos()

self.scrolledtext1.delete("1.0", tk.END)

for fila in respuesta:

self.scrolledtext1.insert(tk.END, "código:"+str(fila[0])+

"\ndescripción:"+fila[1]+

"\nprecio:"+str(fila[2])+"\n\n")

Llamamos al método 'recuperar\_todos' de la clase 'Articulos' y obtenemos una lista con un conjunto de tuplas con cada fila de la tabla.

El algoritmo 'recuperar\_todos' de la clase Articulos es:

def recuperar\_todos(self):

try:

cone=self.abrir()

cursor=cone.cursor()

sql="select codigo, descripcion, precio from articulos"

cursor.execute(sql)

return cursor.fetchall()

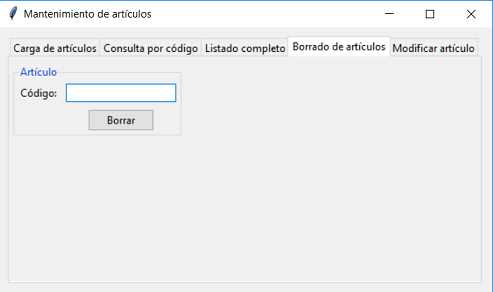
finally:

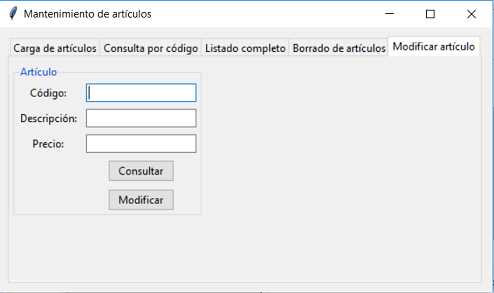
cone.close()

## Problema propuesto

* Agregar dos pestañas al programa de administración de artículos que permitan borrar un artículo ingresando su código y otra opción que permita consultar y modificar la descripción y precio de un artículo.

La interfaces visuales a implementar son:





[Ver video](https://youtu.be/gHYydqF1iiM)

[Solución](javascript:cambiar())

[Retornar](https://www.tutorialesprogramacionya.com/pythonya/index.php?inicio=75)

**módulo: formularioarticulos.py**

import tkinter as tk

from tkinter import ttk

from tkinter import messagebox as mb

from tkinter import scrolledtext as st

import articulos

class FormularioArticulos:

def \_\_init\_\_(self):

self.articulo1=articulos.Articulos()

self.ventana1=tk.Tk()

self.ventana1.title("Mantenimiento de artículos")

self.cuaderno1 = ttk.Notebook(self.ventana1)

self.carga\_articulos()

self.consulta\_por\_codigo()

self.listado\_completo()

self.borrado()

self.modificar()

self.cuaderno1.grid(column=0, row=0, padx=10, pady=10)

self.ventana1.mainloop()

def carga\_articulos(self):

self.pagina1 = ttk.Frame(self.cuaderno1)

self.cuaderno1.add(self.pagina1, text="Carga de artículos")

self.labelframe1=ttk.LabelFrame(self.pagina1, text="Artículo")

self.labelframe1.grid(column=0, row=0, padx=5, pady=10)

self.label1=ttk.Label(self.labelframe1, text="Descripción:")

self.label1.grid(column=0, row=0, padx=4, pady=4)

self.descripcioncarga=tk.StringVar()

self.entrydescripcion=ttk.Entry(self.labelframe1, textvariable=self.descripcioncarga)

self.entrydescripcion.grid(column=1, row=0, padx=4, pady=4)

self.label2=ttk.Label(self.labelframe1, text="Precio:")

self.label2.grid(column=0, row=1, padx=4, pady=4)

self.preciocarga=tk.StringVar()

self.entryprecio=ttk.Entry(self.labelframe1, textvariable=self.preciocarga)

self.entryprecio.grid(column=1, row=1, padx=4, pady=4)

self.boton1=ttk.Button(self.labelframe1, text="Confirmar", command=self.agregar)

self.boton1.grid(column=1, row=2, padx=4, pady=4)

def agregar(self):

datos=(self.descripcioncarga.get(), self.preciocarga.get())

self.articulo1.alta(datos)

mb.showinfo("Información", "Los datos fueron cargados")

self.descripcioncarga.set("")

self.preciocarga.set("")

def consulta\_por\_codigo(self):

self.pagina2 = ttk.Frame(self.cuaderno1)

self.cuaderno1.add(self.pagina2, text="Consulta por código")

self.labelframe2=ttk.LabelFrame(self.pagina2, text="Artículo")

self.labelframe2.grid(column=0, row=0, padx=5, pady=10)

self.label1=ttk.Label(self.labelframe2, text="Código:")

self.label1.grid(column=0, row=0, padx=4, pady=4)

self.codigo=tk.StringVar()

self.entrycodigo=ttk.Entry(self.labelframe2, textvariable=self.codigo)

self.entrycodigo.grid(column=1, row=0, padx=4, pady=4)

self.label2=ttk.Label(self.labelframe2, text="Descripción:")

self.label2.grid(column=0, row=1, padx=4, pady=4)

self.descripcion=tk.StringVar()

self.entrydescripcion=ttk.Entry(self.labelframe2, textvariable=self.descripcion, state="readonly")

self.entrydescripcion.grid(column=1, row=1, padx=4, pady=4)

self.label3=ttk.Label(self.labelframe2, text="Precio:")

self.label3.grid(column=0, row=2, padx=4, pady=4)

self.precio=tk.StringVar()

self.entryprecio=ttk.Entry(self.labelframe2, textvariable=self.precio, state="readonly")

self.entryprecio.grid(column=1, row=2, padx=4, pady=4)

self.boton1=ttk.Button(self.labelframe2, text="Consultar", command=self.consultar)

self.boton1.grid(column=1, row=3, padx=4, pady=4)

def consultar(self):

datos=(self.codigo.get(), )

respuesta=self.articulo1.consulta(datos)

if len(respuesta)>0:

self.descripcion.set(respuesta[0][0])

self.precio.set(respuesta[0][1])

else:

self.descripcion.set('')

self.precio.set('')

mb.showinfo("Información", "No existe un artículo con dicho código")

def listado\_completo(self):

self.pagina3 = ttk.Frame(self.cuaderno1)

self.cuaderno1.add(self.pagina3, text="Listado completo")

self.labelframe3=ttk.LabelFrame(self.pagina3, text="Artículo")

self.labelframe3.grid(column=0, row=0, padx=5, pady=10)

self.boton1=ttk.Button(self.labelframe3, text="Listado completo", command=self.listar)

self.boton1.grid(column=0, row=0, padx=4, pady=4)

self.scrolledtext1=st.ScrolledText(self.labelframe3, width=30, height=10)

self.scrolledtext1.grid(column=0,row=1, padx=10, pady=10)

def listar(self):

respuesta=self.articulo1.recuperar\_todos()

self.scrolledtext1.delete("1.0", tk.END)

for fila in respuesta:

self.scrolledtext1.insert(tk.END, "código:"+str(fila[0])+

"\ndescripción:"+fila[1]+

"\nprecio:"+str(fila[2])+"\n\n")

def borrado(self):

self.pagina4 = ttk.Frame(self.cuaderno1)

self.cuaderno1.add(self.pagina4, text="Borrado de artículos")

self.labelframe4=ttk.LabelFrame(self.pagina4, text="Artículo")

self.labelframe4.grid(column=0, row=0, padx=5, pady=10)

self.label1=ttk.Label(self.labelframe4, text="Código:")

self.label1.grid(column=0, row=0, padx=4, pady=4)

self.codigoborra=tk.StringVar()

self.entryborra=ttk.Entry(self.labelframe4, textvariable=self.codigoborra)

self.entryborra.grid(column=1, row=0, padx=4, pady=4)

self.boton1=ttk.Button(self.labelframe4, text="Borrar", command=self.borrar)

self.boton1.grid(column=1, row=1, padx=4, pady=4)

def borrar(self):

datos=(self.codigoborra.get(), )

cantidad=self.articulo1.baja(datos)

if cantidad==1:

mb.showinfo("Información", "Se borró el artículo con dicho código")

else:

mb.showinfo("Información", "No existe un artículo con dicho código")

def modificar(self):

self.pagina5 = ttk.Frame(self.cuaderno1)

self.cuaderno1.add(self.pagina5, text="Modificar artículo")

self.labelframe5=ttk.LabelFrame(self.pagina5, text="Artículo")

self.labelframe5.grid(column=0, row=0, padx=5, pady=10)

self.label1=ttk.Label(self.labelframe5, text="Código:")

self.label1.grid(column=0, row=0, padx=4, pady=4)

self.codigomod=tk.StringVar()

self.entrycodigo=ttk.Entry(self.labelframe5, textvariable=self.codigomod)

self.entrycodigo.grid(column=1, row=0, padx=4, pady=4)

self.label2=ttk.Label(self.labelframe5, text="Descripción:")

self.label2.grid(column=0, row=1, padx=4, pady=4)

self.descripcionmod=tk.StringVar()

self.entrydescripcion=ttk.Entry(self.labelframe5, textvariable=self.descripcionmod)

self.entrydescripcion.grid(column=1, row=1, padx=4, pady=4)

self.label3=ttk.Label(self.labelframe5, text="Precio:")

self.label3.grid(column=0, row=2, padx=4, pady=4)

self.preciomod=tk.StringVar()

self.entryprecio=ttk.Entry(self.labelframe5, textvariable=self.preciomod)

self.entryprecio.grid(column=1, row=2, padx=4, pady=4)

self.boton1=ttk.Button(self.labelframe5, text="Consultar", command=self.consultar\_mod)

self.boton1.grid(column=1, row=3, padx=4, pady=4)

self.boton1=ttk.Button(self.labelframe5, text="Modificar", command=self.modifica)

self.boton1.grid(column=1, row=4, padx=4, pady=4)

def modifica(self):

datos=(self.descripcionmod.get(), self.preciomod.get(), self.codigomod.get())

cantidad=self.articulo1.modificacion(datos)

if cantidad==1:

mb.showinfo("Información", "Se modificó el artículo")

else:

mb.showinfo("Información", "No existe un artículo con dicho código")

def consultar\_mod(self):

datos=(self.codigomod.get(), )

respuesta=self.articulo1.consulta(datos)

if len(respuesta)>0:

self.descripcionmod.set(respuesta[0][0])

self.preciomod.set(respuesta[0][1])

else:

self.descripcionmod.set('')

self.preciomod.set('')

mb.showinfo("Información", "No existe un artículo con dicho código")

aplicacion1=FormularioArticulos()

**módulo: articulos.py**

import sqlite3

class Articulos:

def abrir(self):

conexion=sqlite3.connect("c:/programaspython/bd1.db")

return conexion

def alta(self, datos):

cone=self.abrir()

cursor=cone.cursor()

sql="insert into articulos(descripcion, precio) values (?,?)"

cursor.execute(sql, datos)

cone.commit()

cone.close()

def consulta(self, datos):

try:

cone=self.abrir()

cursor=cone.cursor()

sql="select descripcion, precio from articulos where codigo=?"

cursor.execute(sql, datos)

return cursor.fetchall()

finally:

cone.close()

def recuperar\_todos(self):

try:

cone=self.abrir()

cursor=cone.cursor()

sql="select codigo, descripcion, precio from articulos"

cursor.execute(sql)

return cursor.fetchall()

finally:

cone.close()

def baja(self, datos):

try:

cone=self.abrir()

cursor=cone.cursor()

sql="delete from articulos where codigo=?"

cursor.execute(sql, datos)

cone.commit()

return cursor.rowcount # retornamos la cantidad de filas borradas

except:

cone.close()

def modificacion(self, datos):

try:

cone=self.abrir()

cursor=cone.cursor()

sql="update articulos set descripcion=?, precio=? where codigo=?"

cursor.execute(sql, datos)

cone.commit()

return cursor.rowcount # retornamos la cantidad de filas modificadas

except:

cone.close()