we design and create professional quality software

Language - Agnostic

Language-agnostic programing

La programación o secuencias de comandos independientes del lenguaje es un paradigma de desarrollo de software en el que se elige un lenguaje en particular debido a su idoneidad para una tarea en particular, y no simplemente por el conjunto de habilidades disponibles dentro de un equipo de desarrollo.

Modern Portfolio Designed

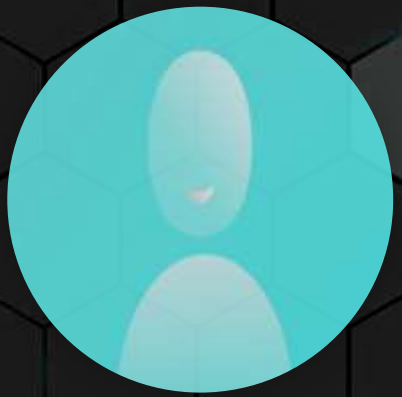
Te quedo? Listo...
(Rodrigo Bueno)

GUI

Python

GUI Frameworks

python



Una GUI es una interfaz de usuario gráfica que utiliza un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz, en lugar de una visualización puramente textual para una computadora.

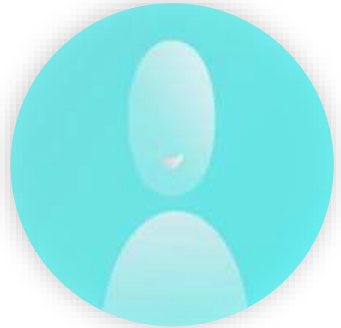
PYTHON

Introducción a GUI

INTRODUCCION GUI en PYTHON.

python

Introducción a GUI



Interfaz gráfica de usuario - GUI

Definición: La Interfaz Gráfica de Usuario, conocida en inglés como Graphical User Interface (GUI) es la forma en que un usuario puede interactuar con un dispositivo informático sin introducir comandos de texto en una consola. Es un entorno visual amigable que permite al usuario realizar cualquier acción sin necesidad de tener conocimientos de programación.

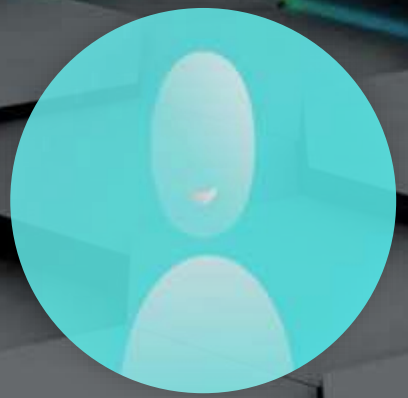
Un ejemplo del GUI son los entornos Windows, Linix, MacOs o Android, gracias a los cuales se pueden enviar comandos a través de gestos o movimientos de ratón, sin necesidad de introducir ningún código.

Evolución de la Interfaz Gráfica de Usuario: Una de las primeras novedades que se integró cuando se empezó a usar la GUI fue el ratón o mousse. Hoy, en sí, es el elemento más corriente que encontramos a la hora de usar un ordenador o dispositivo informático.
En la actualidad los elementos visuales de una interfaz son cada vez más intuitivos. Los diseñadores se esfuerzan en que el usuario pueda ser capaz de hacer uso de ellos sin un aprendizaje previo.

Con el paso del tiempo este tipo de interfaz ha ido evolucionando y los dispositivos que usamos también. Por eso, ahora hemos aumentado de manera exponencial el uso de las conocidos pantallas táctiles o touchscreen. Aquí, los comandos se efectúan al tocar esta pantalla, sin necesidad del ratón. Algunos ejemplos donde también se emplean GUI de uso específico con la pantalla táctil son los cajeros automáticos, las pantallas de monitorización y control en los usos industriales o videoconsolas.

Para qué se usa la Interfaz Gráfica de Usuario El objetivo de la GUI es facilitar la comunicación entre el usuario y el sistema operativo. Cuando la informática empezó a desarrollarse solo se podían usar estos primeros ordenadores si se tenían grandes conocimientos informáticos. Pero a raíz de la expansión de este sector surgió la interfaz. Fue entonces cuando la dificultad de usar los sistemas operativos o elementos informáticos se redujo notablemente. Una de las empresas pioneras en este campo fue Apple.

Elementos debe tener una buena Interfaz Gráfica de Usuario: Para que una Interfaz Gráfica de Usuario tenga éxito y se pueda usar fácilmente debe cumplir una serie de requisitos. Lo primero es que sea sencilla de entender. La llamada curva de aprendizaje debe ser rápida. Los elementos principales deben ser también muy identificables. Par ello es importante facilitar y predecir las acciones más comunes de un usuario. La información debe estar ordenada mediante menús, iconos, imágenes… Así será intuitiva y las operaciones para hacer y deshacer se podrán realizar de forma rápida. La usabilidad debe ser fácil.
.



Python

GUI Frameworks

Una GUI o una interfaz gráfica de usuario es un entorno interactivo para tomar respuestas de los usuarios en diversas situaciones, como formularios, documentos, pruebas, etc. Proporciona al usuario una buena pantalla interactiva que una interfaz de línea de comandos (CLI) tradicional.

python

Tk&

kivy

PYQT5

PySide
Python for Qt

wxPython

Bibliotecas de GUI de Python

1. PyQT5

PyQT5 es un marco de interfaz gráfica de usuario (GUI) para Python. Es muy popular entre los desarrolladores y la GUI se puede crear mediante codificación o un diseñador QT. Un marco de desarrollo QT es un marco visual que permite arrastrar y soltar widgets para construir interfaces de usuario.

Es un software de enlace gratuito y de código abierto y se implementa para el marco de desarrollo de aplicaciones multiplataforma. Se utiliza en Windows, Mac, Android, Linux y Raspberry PI.

Para la instalación de PyQT5, puede utilizar el siguiente comando:

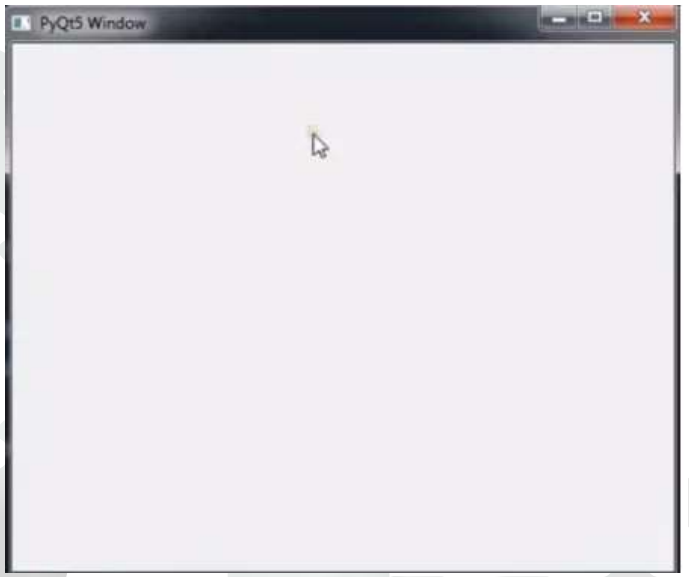
pip install pyqt5

Aquí se muestra un código simple:

Output

```
1  from PyQt5.QtWidgets import QApplication,
       QMainWindow
2  import sys
3
4
5  class Window(QMainWindow):
6      def __init__(self):
7          super().__init__()
8
9          self.setGeometry(300, 300, 600, 400)
10         self.setWindowTitle("PyQt5 window")
11         self.show()
12
13
14 app = QApplication(sys.argv)
15 window = Window()
16
```

Ln: 16 Col: 0 size: 318 B



El resultado del código anterior es el siguiente:

El equipo de desarrolladores de Python de ScienceSoft destaca los beneficios de usar PyQt:

PyQt es un conjunto maduro de enlaces de Python a Qt para el desarrollo multiplataforma de aplicaciones de escritorio. Ofrece una amplia selección de widgets y herramientas integradas para la creación de widgets personalizados para dar forma a GUI sofisticadas, así como un sólido soporte de base de datos SQL para conectarse e interactuar con bases de datos.

Bibliotecas de GUI de Python

2. Python Tkinter

Otro marco GUI se llama Tkinter. Tkinter es una de las bibliotecas GUI de Python más populares para desarrollar aplicaciones de escritorio. Es una combinación del marco GUI estándar de TK y python.

Tkinter proporciona diversos widgets como etiquetas, botones, cuadros de texto, casillas de verificación que se utilizan en una aplicación de interfaz gráfica de usuario..

Los widgets de control de botones se utilizan para mostrar y desarrollar aplicaciones, mientras que el widget de lienzo se utiliza para dibujar formas como líneas, polígonos, rectángulos, etc. en la aplicación. Además, Tkinter es una biblioteca incorporada para Python, por lo que no necesita instalarla como otro marco de GUI. A

A continuación se muestra un ejemplo de codificación usando Tkinter.

Output

```
1  from tkinter import *
2
3
4  class Root(Tk):
5      def __init__(self):
6          super(Root, self).__init__()
7
8          self.title("Python Tkinter")
9          self.minsize(500, 400)
10
11
12 root = Root()
13 root.mainloop()
14
```

Ln: 14 Col: 0 size: 240 B

El resultado del código anterior es el siguiente:

Bibliotecas de GUI de Python

3. PySide 2

La tercera biblioteca GUI de Python de la que vamos a hablar es PySide2 o puede llamarla QT para python. Qt for Python ofrece los enlaces oficiales de Python para Qt (PySide2), lo que permite el uso de sus API en aplicaciones Python, y una herramienta generadora de enlaces (Shiboken2)
que se puede utilizar para exponer proyectos de C ++ en Python.

Qt para Python está disponible bajo la licencia LGPLv3/GPLv3 y la licencia comercial Qt.
Así que ahora déjame mostrarte el proceso de instalación y también un ejemplo. Entonces, para la instalación, simplemente puede usar:

pip install PySide2

Aquí, hay un ejemplo para configurar el marco GUI usando PySide2.

Output

```
1  from PySide2.QtWidgets import QtWidgets,
       QApplication
2  import sys
3
4
5  class Window(QtWidgets):
6      def __init__(self):
7          super().__init__()
8
9          self.setWindowTitle("Pyside2 Window")
10         self.setGeometry(300, 300, 500, 400)
11
12
13 app = QApplication(sys.argv)
14 window = Window()
15 window.show()
16 app.exec_()
17
```

Ln: 17 Col: 0 size: **357** B

El resultado del código anterior es el siguiente:

Bibliotecas de GUI de Python

4. Kivy

Otro marco GUI del que vamos a hablar se llama Kivy. Kivy es una biblioteca Python de código abierto para el desarrollo rápido de aplicaciones que hacen uso de interfaces de usuario innovadoras, como aplicaciones multitáctiles, Kivy se ejecuta en Linux, Windows, OS X, Android, iOS y Raspberry Pi. Puede ejecutar el mismo código en todas las plataformas compatibles. Puede utilizar de forma nativa la mayoría de las entradas, protocolos y dispositivos, incluidos WM_Touch, WM_Pen, Mac OS X Trackpad y Magic Mouse, Mtdev, Linux Kernel HID. Kivy es 100% de uso gratuito, bajo una licencia MIT.

El kit de herramientas se desarrolla, respalda y utiliza profesionalmente. Puedes usarlo en un producto comercial. El marco es estable y tiene una API bien documentada, además de una guía de programación para ayudarlo a comenzar.
El motor gráfico de Kivy está construido sobre OpenGL ES 2, utilizando una tubería de gráficos moderna y rápida.
El kit de herramientas viene con más de 20 widgets, todos altamente extensibles. Muchas partes están escritas en C usando Cython y probadas con pruebas de regresión.
Al llegar a la instalación de Kivy, debe instalar la dependencia "glew". Puede usar el comando pip de la siguiente manera:

pip install docutils pygments
pypiwin32 kivy.deps.sdl2 kivy.deps.Glew

Ingrese este comando y presione enter, se instalará. Después de eso, debe escribir este comando para instalar Kivy:

pip install Kivy

Entonces, después de la instalación, permítanme mostrarles un ejemplo simple de Kivy para mostrar lo fácil que es.

Output

```
1  from kivy.app import App
2  from kivy.uix.button import Button
3
4
5  class TestApp(App):
6      def build(self):
7          return Button(text=" Hello Kivy World ")
8
9
10 TestApp().run()
11
```

Ln: 11 Col: 0 size: 199 B

El resultado del código anterior es el siguiente:

Bibliotecas de GUI de Python

5. wxPython

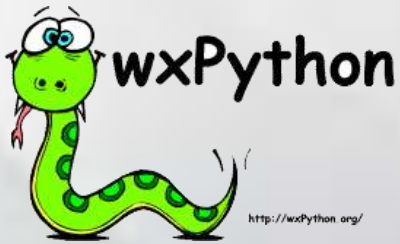
Así que el último marco GUI del que vamos a hablar es wxPython. wxPython es un kit de herramientas GUI multiplataforma para el lenguaje de programación Python.
Permite a los programadores de Python crear programas con una interfaz gráfica de usuario robusta y altamente funcional, de manera simple y sencilla. Se implementa como un conjunto de módulos de extensión de Python que envuelven los componentes de la GUI de la popular biblioteca multiplataforma wxWidgets, que está escrita en C ++.

Al igual que Python y wxWidgets, wxPython es de código abierto.

wxPython es un kit de herramientas multiplataforma. Esto significa que el mismo programa se ejecutará en múltiples plataformas sin modificaciones. Actualmente, las plataformas compatibles son Microsoft Windows, Mac OS X y macOS, y Linux.

Ahora voy a mostrarte el proceso de instalación y crear un ejemplo simple. Entonces, para la instalación, simplemente escriba el siguiente comando:

pip install wxPython
Aquí hay un ejemplo:



Output

```
1  import wx
2  class MyFrame(wx.Frame):
3      def __init__(self, parent, title):
4          super(MyFrame, self).__init__(parent,
               title=title, size=(400, 300))
5      self.panel = MyPanel(self)
6
7  class MyPanel(wx.Panel):
8      def __init__(self, parent):
9          super(MyPanel, self).__init__(parent)
10
11 class MyApp(wx.App):
12     def OnInit(self):
13         self.frame = MyFrame(parent=None, title
               ="wxPython Window")
14         self.frame.show()
15         return True
16
17 app = MyApp()
18 app.MainLoop()
```

Ln: 17 Col: 0 size: **525** B

Bibliotecas de GUI de Python

Conclusión

Así que ahora hemos visto 5 bibliotecas GUI de Python y, en mi opinión, PySide2 y pyQt5 son los marcos GUI más potentes. Pero vienen con una licencia comercial y eso explica por qué son ricos en funciones. Tkinter, Kivy y wxPython son las bibliotecas GUI gratuitas para Python.

kivy

PYTHON GUI
FRAMEWORKS
FOR DEVELOPERS

wxPython

PySide
Python for Qt

PYQT5

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 de escritorio con Python

PYTHON

Introducción a Tcl/Tk (tkinter)

Event Driven

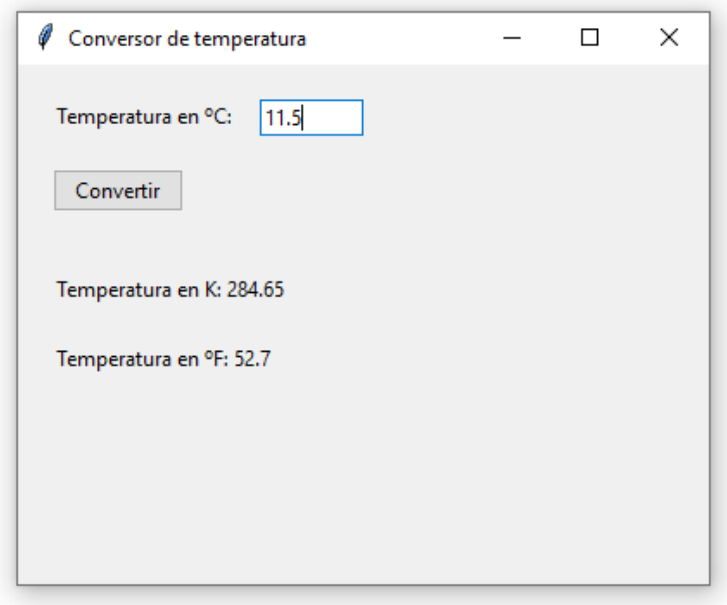
python

Introducción a Tcl/Tk (tkinter)

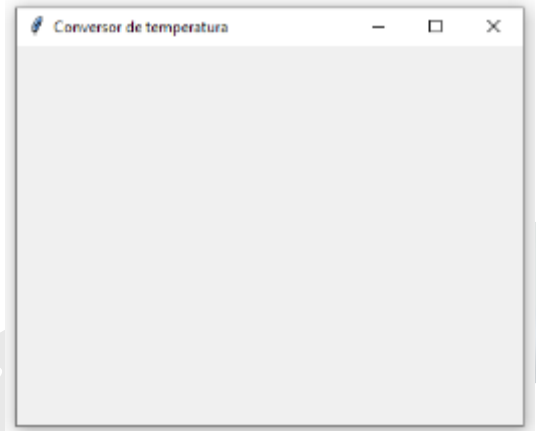
Tk es una herramienta para desarrollar aplicaciones de escritorio multiplataforma, esto es, aplicaciones nativas con una interfaz gráfica para sistemas operativos Windows, Linux, Mac y otros. Técnicamente, Tk es una biblioteca de código abierto escrita en C y desarrollada en sus orígenes para el lenguaje 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, que se integran muy bien con Tk.

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.

.




```
import tkinter as tk
from tkinter import ttk

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



Acceso al IDLE

En informática, un widget o artilugio es una pequeña aplicación o programa, usualmente presentado en clases, archivos o ficheros pequeños que son ejecutados por un motor de widgets o Widget Engine. Entre sus objetivos están dar fácil acceso a funciones frecuentemente usadas y proveer de información visual.

PYTHON

Widgets

Tcl/Tk (tkinter)

Event Driven

python

Introducción a Tcl/Tk (tkinter) Widgets

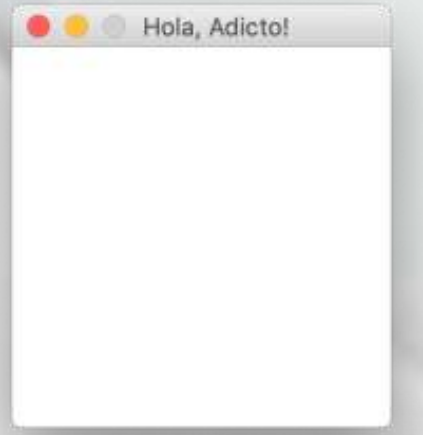
1. Introducción

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

Widgets

A la hora de montar una vista con Tkinter, nos basaremos en widgets jerarquizados, que irán componiendo poco a poco nuestra interfaz. Algunos de los más comunes son:

1. Tk: es la raíz de la interfaz, donde vamos a colocar el resto de widgets.

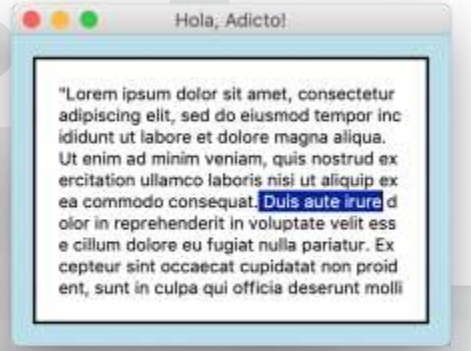


2. Frame: marco que permite agrupar diferentes widgets.

3. Label: etiqueta estática que permite mostrar texto o imagen.

4. Entry: etiqueta que permite introducir texto corto (típico de formularios).

5. Text: campo que permite introducir texto largo (típico para añadir comentarios).



•*6. Button*: ejecuta una función al ser pulsado.

python

Introducción a Tcl/Tk (tkinter)

Widgets

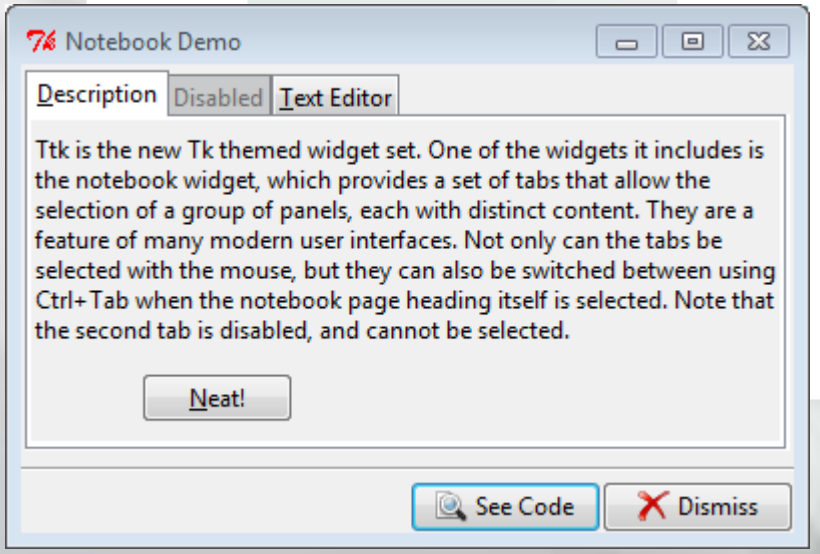
A la hora de montar una vista con Tkinter, nos basaremos en widgets jerarquizados, que irán componiendo poco a poco nuestra interfaz. Algunos de los más comunes son:

7, Radiobutton: permite elegir una opción entre varias.

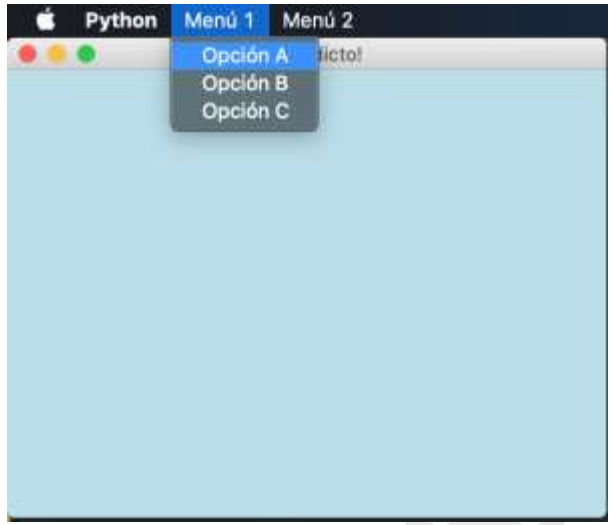
8, Checkbutton: permite elegir varias de las opciones propuestas.

10, Dialogs: ventana emergente (o pop-up).

11,Notebook

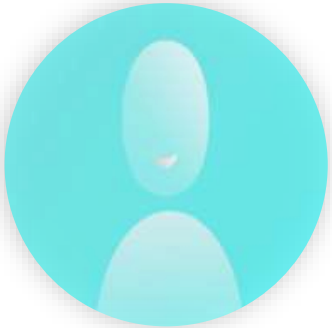


9. Menu: clásico menú superior con opciones (Archivo, Editar…).



Cuando vayamos a inicializar el componente, debemos pasar por constructor el elemento que quede “por encima” en la jerarquía de la vista (si queremos colocar una label dentro de un frame, al construir la etiqueta le pasaremos el marco como argumento del constructor).

Introducción a Tcl/Tk (tkinter)



Configuración

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

- bg: modifica el color de fondo. Se puede indicar con el color en inglés (incluyendo modificadores, como “darkgreen”) o su código RGB en hexadecimal (“#aaaaaa” para blanco). Ojo: en MacOS no se puede modificar el color de fondo de los botones; aunque indiquemos un nuevo color, se mostrará en blanco. Lo más parecido que podemos hacer es configurar el highlightbackground, que pintará el fondo alrededor del botón del color que indiquemos.
- fg: cambia el color del texto.
- cursor: modifica la forma del cursor. Algunos de los más utilizados son “gumby”, “pencil”, “watch” o “cross”.
- height: altura en líneas del componente.
- width: anchura en caracteres del componente.
- font: nos permite especificar, en una tupla con nombre de la fuente, tamaño y estilo, la fuente a utilizar en el texto del componente. Por ejemplo, Font(“Times New Roman”, 24, “bold underline”).
- bd: modificamos la anchura del borde del widget.
- relief: cambiamos el estilo del borde del componente. Su valor puede ser “flat”, “sunken”, “raised”, “groove”, “solid” o “ridge”.
- state: permite deshabilitar el componente (state=DISABLED); por ejemplo, una Label en la que no se puede escribir o un Button que no se puede clickar.
- padding: espacio en blanco alrededor del widget en cuestión.
- command: de cara a que los botones hagan cosas, podemos indicar qué función ejecutar cuando se haga click en el mismo.

Python 3.5.1 Shell

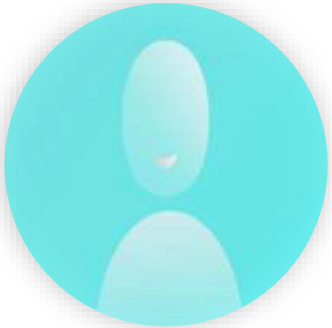
File Edit Shell Debug Options Window Help

```
Python 3.5.1 (default, Dec  7 2015, 12:58:09)
[GCC 5.2.0] on linux
Type "copyright", "credits" or "license()" for more information.
>>> from six.moves import tkinter as tk
>>> def funcion():
    print("Excelente")


>>> root = tk.Tk()
>>> boton = tk.Button(root, text="Que te parece la guía?", command=funcion)
>>> boton.pack()
>>> root.mainloop()
Excelente
```

Ln: 9 Col: 18

python

Introducción a Tcl/Tk (tkinter)



Gestión de la composición

Es MUY IMPORTANTE que, cuando tengamos configurado el componente, utilicemos un gestor de geometría de componentes. Si no, el widget quedará creado pero no se mostrará.

Los tres más conocidos son:

- Pack: cuando añadimos un nuevo componente, se “hace hueco” a continuación de los que ya están incluidos (podemos indicar que se inserte en cualquiera de las 4 direcciones), para finalmente calcular el tamaño que necesita el widget padre para contenerlos a todos.
- Place: este es el más sencillo de entender, pero puede que no el más sencillo de utilizar para todo el mundo. Al insertar un componente, podemos indicar explícitamente la posición (coordenadas X e Y) dentro del widget padre, ya sea en términos absolutos o relativos.
- Grid: la disposición de los elementos es una matriz, de manera que para cada uno debemos indicar la celda (fila y columna) que queremos que ocupe. Podemos además especificar que ocupe más de una fila y/o columna (rowspan/columnspan=3), “pegarlo” a cualquiera de los 4 bordes de la celda en vez de centrarlo (sticky=W para el borde izquierdo, por ejemplo)…

Personalmente, me siento mucho más cómodo utilizando un Grid: de esta manera te aseguras la distribución de los componentes, y la hora de añadir nuevos es muy visual poder pensar en ello como matriz, en vez de coordenadas o posiciones relativas a otros widgets.

Ejecución

Una vez tenemos todos los componentes creados, configurados y añadidos en la estructura, debemos terminar el script con la instrucción tk.mainloop() (“tk” = variable Tk). Así, cuando lo ejecutemos, se abrirá la ventana principal de nuestra GUI.

Tip: si guardamos nuestro script con formato .pyw en vez de .py, al ejecutarlo se abrirá nuestra interfaz, sin tener que pasar por terminal o abrir algún IDE para ello.

Python ofrece múltiples opciones para desarrollar GUI (interfaz gráfica de usuario). De todos los métodos GUI, tkinter es el método más utilizado. Es una interfaz Python estándar para el kit de herramientas Tk GUI enviado con Python. Python con tkinter es la forma más rápida y fácil de crear las aplicaciones GUI. Crear una GUI usando tkinter es una tarea fácil..

PYTHON

Tcl/Tk (tkinter)

Coding...

Event Driven

python

Introducción a Tcl/Tk (tkinter)

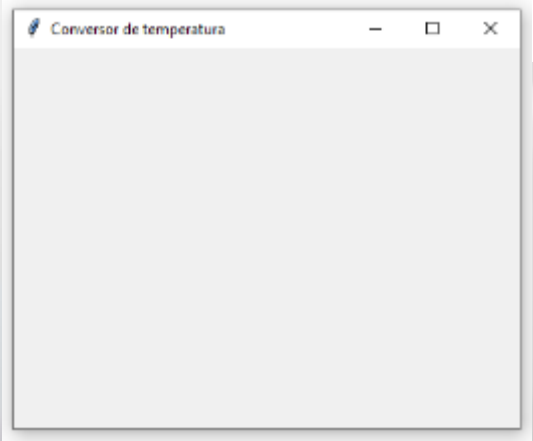
Para comenzar, el primer paso para cualquier aplicación de Tk es importar los módulos correspondientes:

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. Puede ampliar en la Web las diferencias entre tk y ttk.

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

3. ventana = tk.Tk()
4. 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().

```
import tkinter as tk
from tkinter import ttk

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

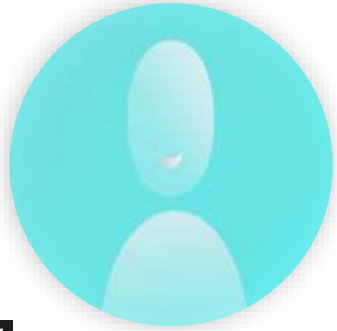
En cualquier método, función o clase de Tk donde se requiera especificar un tamaño, será vía los argumentos width (ancho) y height (alto), indicados en píxeles. Así, nuestra ventana tendrá al iniciarse un ancho de 400 píxeles y un alto de 300 píxeles.

0 Py

Acceso al IDLE

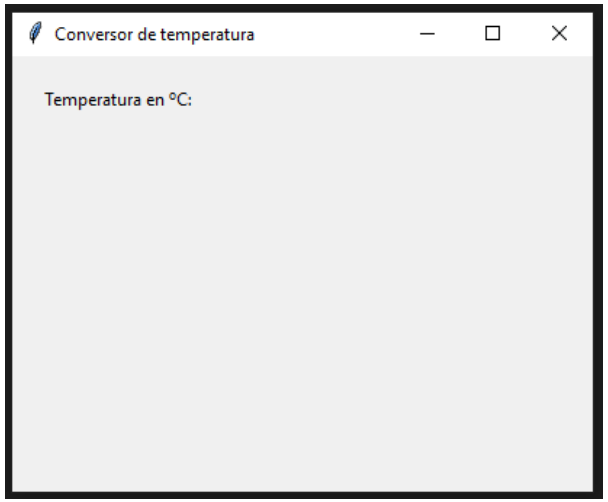
python

Introducción a Tcl/Tk (tkinter)



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. 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

```
import tkinter as tk
from tkinter import ttk
ventana = tk.Tk()
ventana.title("Conversor de temperatura")
ventana.config(width=400, height=300)
etiqueta_temp_celsius = ttk.Label(text="Temperatura en ºC:")
etiqueta_temp_celsius.place(x=20, y=20)
ventana.mainloop()
```

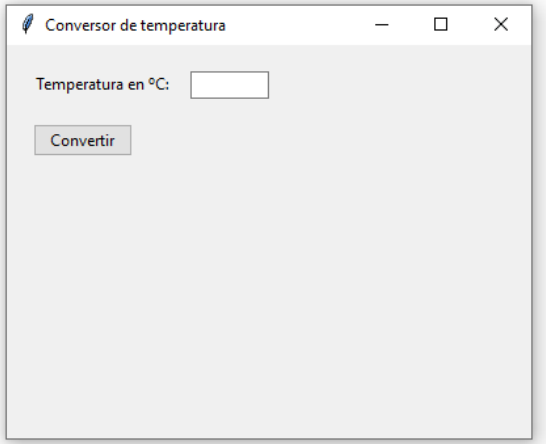
Con Orientación a Objetos

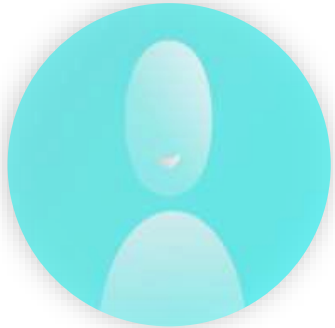
```
import tkinter as tk
from tkinter import ttk
class Aplicacion(ttk.Frame):
    def __init__(self, parent):
        super().__init__(parent)
        self.etiqueta_temp_celsius = ttk.Label(parent, text="Temperatura en ºC:")
        self.etiqueta_temp_celsius.place(x=20, y=20)
ventana = tk.Tk()
ventana.title("Conversor de temperatura")
ventana.config(width=400, height=300)
app = Aplicacion(ventana)
ventana.mainloop()
```

python

Introducción a Tcl/Tk (tkinter)

Introducir Controles a la Interfaz





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.

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

```
import tkinter as tk
from tkinter import ttk
ventana = tk.Tk()
ventana.title("Conversor de temperatura")
ventana.config(width=400, height=300)
etiqueta_temp_celsius = ttk.Label(text="Temperatura en ºC:")
etiqueta_temp_celsius.place(x=20, y=20)
caja_temp_celsius = ttk.Entry()
caja_temp_celsius.place(x=140, y=20, width=60)
boton_convertir = ttk.Button(text="Convertir")
boton_convertir.place(x=20, y=60)
ventana.mainloop()
```

Con Orientación a Objetos

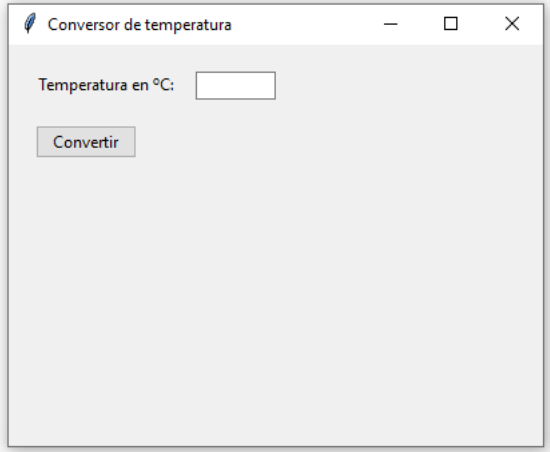
```
import tkinter as tk
from tkinter import ttk
class Aplicacion(ttk.Frame):
    def __init__(self, parent):
        super().__init__(parent)
        self.etiqueta_temp_celsius = ttk.Label(
        parent, text="Temperatura en ºC:")
        self.etiqueta_temp_celsius.place(x=20, y=20)
        self.caja_temp_celsius = ttk.Entry(parent)
        self.caja_temp_celsius.place(x=140, y=20, width=60)
        self.boton_convertir = ttk.Button(
        parent, text="Convertir")
        self.boton_convertir.place(x=20, y=60)

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

Acceso al IDLE

python

Introducción a Tcl/Tk (tkinter)

Introducir Controles a la Interfaz 2



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

```
import tkinter as tk
from tkinter import ttk
ventana = tk.Tk()
ventana.title("Conversor de temperatura")
ventana.config(width=400, height=300)
etiqueta_temp_celsius = ttk.Label(text="Temperatura en ºC:")
etiqueta_temp_celsius.place(x=20, y=20)
caja_temp_celsius = ttk.Entry()
caja_temp_celsius.place(x=140, y=20, width=60)
boton_convertir = ttk.Button(text="Convertir")
boton_convertir.place(x=20, y=60)
etiqueta_temp_kelvin = ttk.Label(text="Temperatura en K: n/a")
etiqueta_temp_kelvin.place(x=20, y=120)
etiqueta_temp_fahrenheit = ttk.Label(text="Temperatura en ºF: n/a")
etiqueta_temp_fahrenheit.place(x=20, y=160)
ventana.mainloop()
```

Con Orientación a Objetos

```
import tkinter as tk
from tkinter import ttk
class Aplicacion(ttk.Frame):
    def __init__(self, parent):
        super().__init__(parent)
        self.etiqueta_temp_celsius = ttk.Label(
        parent, text="Temperatura en ºC:")
        self.etiqueta_temp_celsius.place(x=20, y=20)
        self.caja_temp_celsius = ttk.Entry(parent)
        self.caja_temp_celsius.place(x=140, y=20, width=60)
        self.boton_convertir = ttk.Button(
        parent, text="Convertir")
        self.boton_convertir.place(x=20, y=60)

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

Acceso al IDLE

python

Un evento es una señal que comunica a una aplicación que ha sucedido algo importante. Por ejemplo, cuando un usuario hace clic en un control en un formulario, el formulario puede provocar un evento Click y llamar a un procedimiento que controla el evento.

PYTHON

Manejo de Eventos Tcl/Tk (tkinter)

Event Driven

python

Introducción a Tcl/Tk (tkinter)

Eventos en la Interfaz

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

```
import tkinter as tk
from tkinter import ttk
def convertir_temp():
      temp_celsius = float(caja_temp_celsius.get())
      temp_kelvin = temp_celsius + 273.15
      temp_fahrenheit = temp_celsius*1.8 + 32
      etiqueta_temp_kelvin.config(text=f"Temperatura en K:
      {temp_kelvin}")
      etiqueta_temp_fahrenheit.config(
      text=f"Temperatura en ºF: {temp_fahrenheit}")
ventana = tk.Tk()
ventana.title("Conversor de temperatura")
ventana.config(width=400, height=300)
etiqueta_temp_celsius = ttk.Label(text="Temperatura en ºC:")
etiqueta_temp_celsius.place(x=20, y=20)
caja_temp_celsius = ttk.Entry()
caja_temp_celsius.place(x=140, y=20, width=60)
boton_convertir = ttk.Button(text="Convertir", command=convertir_temp)
boton_convertir.place(x=20, y=60)
etiqueta_temp_kelvin = ttk.Label(text="Temperatura en K: n/a")
etiqueta_temp_kelvin.place(x=20, y=120)
etiqueta_temp_fahrenheit = ttk.Label(text="Temperatura en ºF: n/a")
etiqueta_temp_fahrenheit.place(x=20, y=160)
ventana.mainloop()
```

Con Orientación a Objetos

```
import tkinter as tk
from tkinter import ttk
class Aplicacion(ttk.Frame):
      def __init__(self, parent):
         super().__init__(parent)
         self.etiqueta_temp_celsius = ttk.Label(
         parent, text="Temperatura en ºC:")
         self.etiqueta_temp_celsius.place(x=20, y=20)
         self.caja_temp_celsius = ttk.Entry(parent)
         self.caja_temp_celsius.place(x=140, y=20, width=60)
         self.boton_convertir = ttk.Button(
         parent, text="Convertir", command=self.convertir_temp)
         self.boton_convertir.place(x=20, y=60)
         self.etiqueta_temp_kelvin = ttk.Label(
         parent, text="Temperatura en K: n/a")
         self.etiqueta_temp_kelvin.place(x=20, y=120)
         self.etiqueta_temp_fahrenheit = ttk.Label(
         parent, text="Temperatura en ºF: n/a")
         self.etiqueta_temp_fahrenheit.place(x=20, y=160)
      def convertir_temp(self):
         temp_celsius = float(self.caja_temp_celsius.get())
         temp_kelvin = temp_celsius + 273.15
         temp_fahrenheit = temp_celsius*1.8 + 32
         self.etiqueta_temp_kelvin.config(
         text=f"Temperatura en K: {temp_kelvin}")
         self.etiqueta_temp_fahrenheit.config(
         text=f"Temperatura en ºF: {temp_fahrenheit}")

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

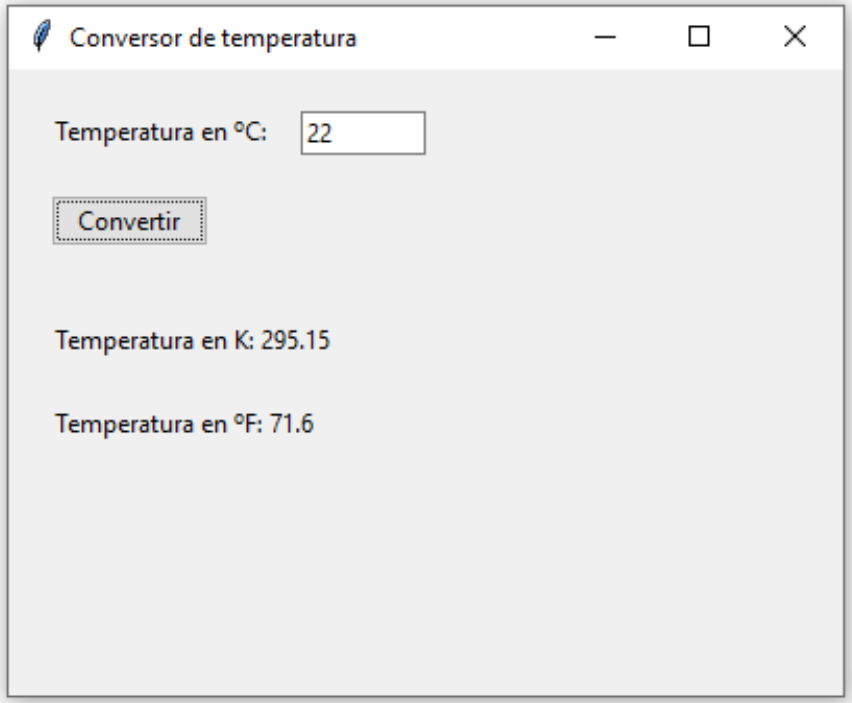
Introducción a Tcl/Tk (tkinter)

Eventos en la Interfaz

¡Excelente! La función convertir_temp() será invocada por Tk cada vez que el usuario presione el boton_convertir.. El contenido de la caja de texto hemos tenido que convertirlo a un número de coma flotante vía la función incorporada float(), ya que por defecto el resultado de get() es siempre una cadena. En hora buena !!!, una pequeña aplicación de escritorio escrita con Python y Tk y sus conceptos principales: ventana, controles, bucle principal, etc.

Sin Orientación a Objetos

```
import tkinter as tk
from tkinter import ttk

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

ventana = tk.Tk()
ventana.title("Conversor de temperatura")
ventana.config(width=400, height=300)
etiqueta_temp_celsius = ttk.Label(text="Temperatura en ºC:")
etiqueta_temp_celsius.place(x=20, y=20)
caja_temp_celsius = ttk.Entry()
caja_temp_celsius.place(x=140, y=20, width=60)
boton_convertir = ttk.Button(text="Convertir", command=convertir_temp)
boton_convertir.place(x=20, y=60)
etiqueta_temp_kelvin = ttk.Label(text="Temperatura en K: n/a")
etiqueta_temp_kelvin.place(x=20, y=120)
etiqueta_temp_fahrenheit = ttk.Label(text="Temperatura en ºF: n/a")
etiqueta_temp_fahrenheit.place(x=20, y=160)
ventana.mainloop()
```

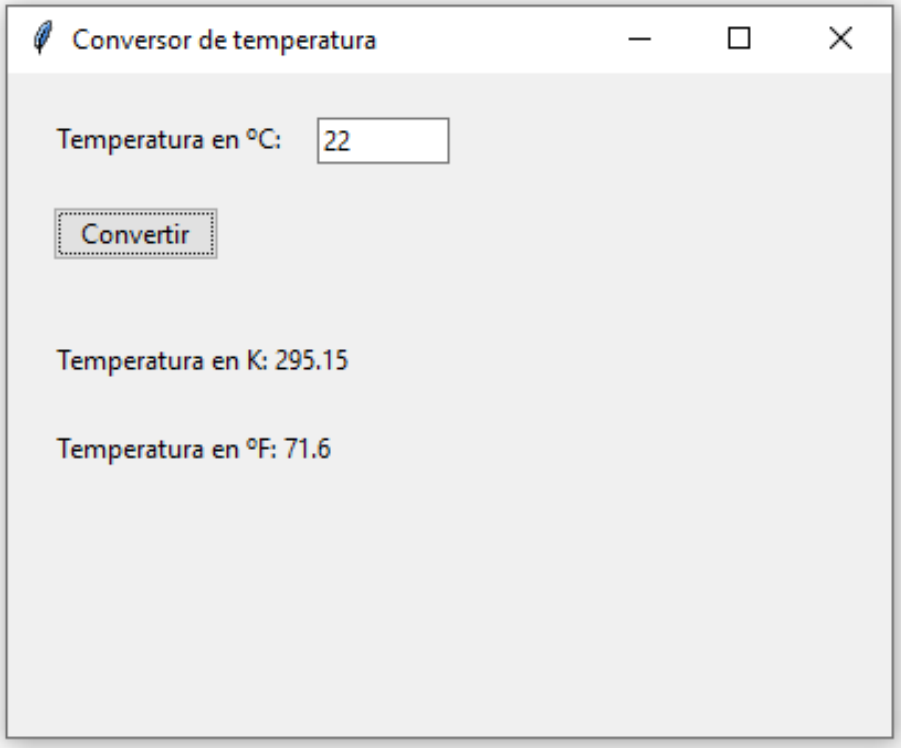


Acceso al IDLE

Introducción a Tcl/Tk (tkinter)

Eventos en la Interfaz

¡Excelente! La función convertir_temp() será invocada por Tk cada vez que el usuario presione el boton_convertir.. El contenido de la caja de texto hemos tenido que convertirlo a un número de coma flotante vía la función incorporada float(), ya que por defecto el resultado de get() es siempre una cadena. Sobre las cajas de texto tenemos este otro artículo. He aquí entonces una pequeña aplicación de escritorio escrita con Python y Tk y sus conceptos principales: ventana, controles, bucle principal, etc.



Para más detalles sobre el funcionamiento de los botones, véase este artículo

Con Orientación a Objetos

```python
import tkinter as tk
from tkinter import ttk
class Aplicacion(ttk.Frame):
    def __init__(self, parent):
        super().__init__(parent)
        self.etiqueta_temp_celsius = ttk.Label(
        parent, text="Temperatura en ºC:")
        self.etiqueta_temp_celsius.place(x=20, y=20)
        self.caja_temp_celsius = ttk.Entry(parent)
        self.caja_temp_celsius.place(x=140, y=20, width=60)
        self.boton_convertir = ttk.Button(
        parent, text="Convertir", command=self.convertir_temp)
        self.boton_convertir.place(x=20, y=60)
        self.etiqueta_temp_kelvin = ttk.Label(
        parent, text="Temperatura en K: n/a")
        self.etiqueta_temp_kelvin.place(x=20, y=120)
        self.etiqueta_temp_fahrenheit = ttk.Label(
        parent, text="Temperatura en ºF: n/a")
        self.etiqueta_temp_fahrenheit.place(x=20, y=160)
    def convertir_temp(self):
        temp_celsius = float(self.caja_temp_celsius.get())
        temp_kelvin = temp_celsius + 273.15
        temp_fahrenheit = temp_celsius*1.8 + 32
        self.etiqueta_temp_kelvin.config(
        text=f"Temperatura en K: {temp_kelvin}")
        self.etiqueta_temp_fahrenheit.config(
        text=f"Temperatura en ºF: {temp_fahrenheit}")

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

TALK IS CHEAP SHOW ME THE CODE

02 GUI Py

python

Pack:
cuando añadimos un nuevo componente, se “hace hueco” a continuación de los que ya están incluidos

Place:
Al insertar un componente, podemos indicar explícitamente la posición (coordenadas X e Y) dentro del widget padre, ya sea en términos absolutos o relativos.

Grid:
la disposición de los elementos es una matriz, de manera que para cada uno debemos indicar la celda (fila y columna) que queremos que ocupe.

PYTHON

Posicionar elementos en Tcl/Tk (tkinter)

Place – Pack - Grid

python

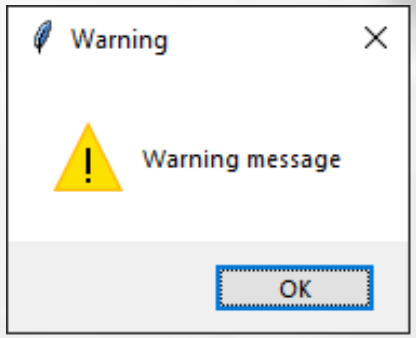
Introducción a Tcl/Tk (tkinter)

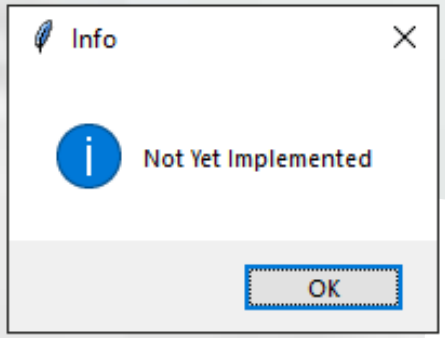
Posicionar elementos en Tcl/Tk (tkinter)

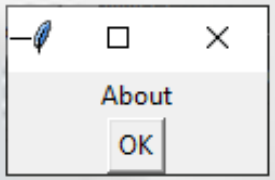
Tk provee tres métodos para establecer la posición de los controles o widgets dentro de una ventana, que se corresponden con las funciones **pack(), place() y grid()**. Algunos son más versátiles, otros más restrictivos. ¿Cuál debes usar? Dependerá del resultado que se quiera lograrr. Explicaremos cada uno de ellos.

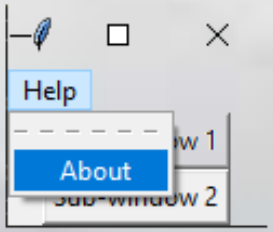
Nota: no deben mezclarse distintos métodos dentro de una misma aplicación.

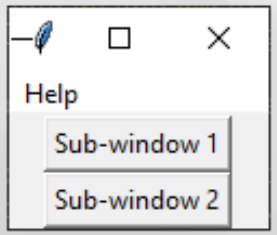
pack(), place() y grid()

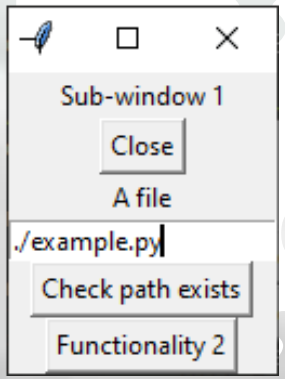


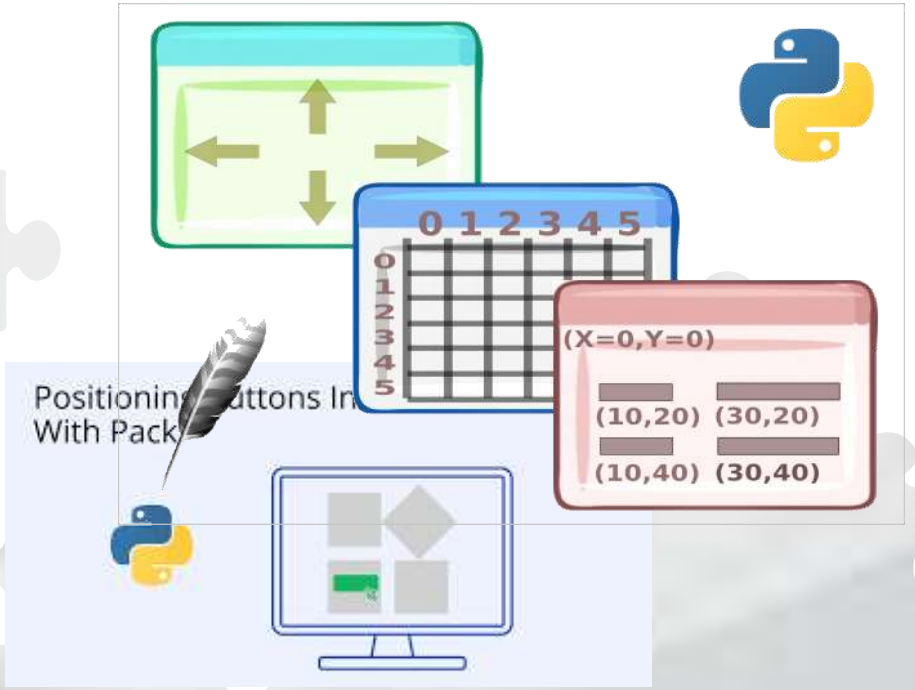












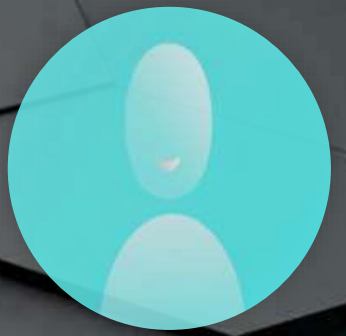
Acceso al IDLE

python

Place: este es el más sencillo de entender, pero puede que no el más sencillo de utilizar para todo el mundo. Al insertar un componente, podemos indicar explícitamente la posición (coordenadas X e Y) dentro del widget padre, ya sea en términos absolutos o relativos.

PYTHON place

Posicionar elementos en Tcl/Tk (tkinter)



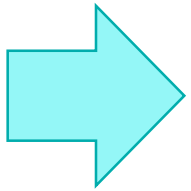
python

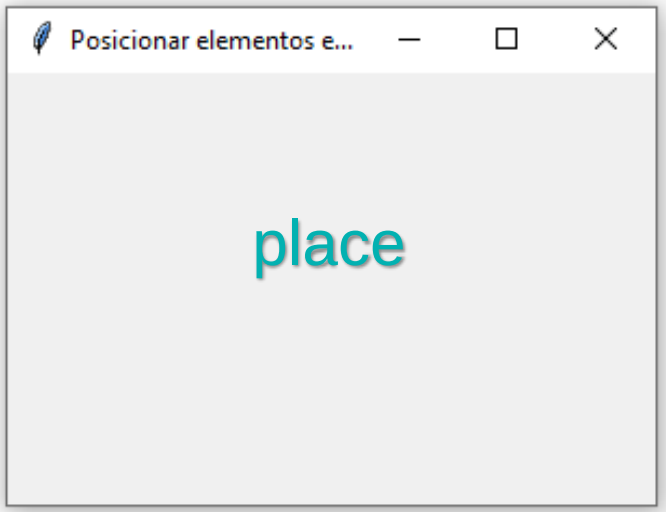
Introducción a Tcl/Tk (tkinter)

Posicionar elementos en Tcl/Tk (tkinter) **Posición absoluta (place)**

La función place() permite ubicar elementos indicando su posición (X e Y) respecto de un elemento padre. En general casi todas las librerías gráficas proveen una opción de este tipo, ya que es la más intuitiva. Para ver un ejemplo, consideremos el siguiente código.

```
# -*- coding: utf-8 -*-
import tkinter as tk
from tkinter import ttk
class Application(ttk.Frame):
    def __init__(self, main_window):
        super().__init__(main_window)
        main_window.title("Posicionar elementos en Tcl/Tk")
        main_window.configure(width=300, height=200)
        # Ignorar esto por el momento.
        self.place(relwidth=1, relheight=1)
main_window = tk.Tk()
app = Application(main_window)
app.mainloop()
```





Este pequeño programa simplemente crea una ventana (main_window) con un widget padre (Application que hereda de ttk.Frame) que contentrá al resto de los elementos. Tanto la ventana principal como el elemento padre tienen un tamaño de 300×200 píxeles.

Ahora bien, vamos a crear un botón y lo vamos a ubicar en la posición (60, 40).
Cabe aclarar que no deben mezclarse distintos métodos dentro de una misma aplicación.

Acceso al IDLE

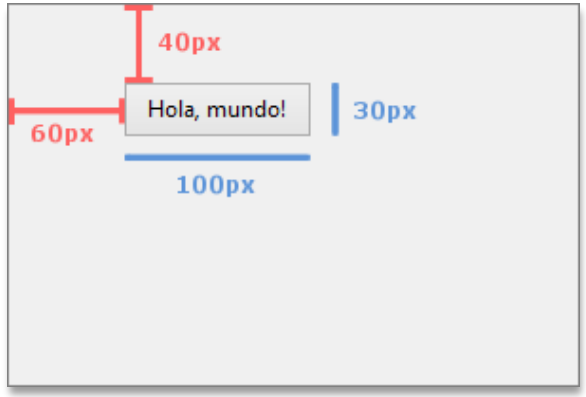
python

Introducción a Tcl/Tk (tkinter)

Posicionar elementos en Tcl/Tk (tkinter) **Posición absoluta (place)**

Ahora bien, vamos a crear un botón y lo vamos a ubicar en la posición (60, 40).

```
        self.button = ttk.Button(self, text="Hola, mundo!")
        self.button.place(x=60, y=40)
```



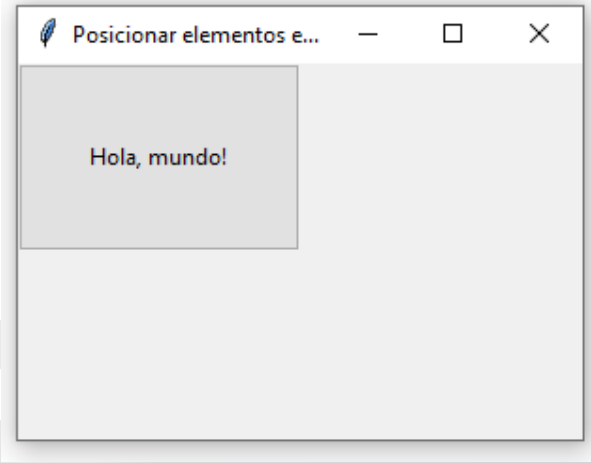
Ya que el origen de coordenadas (es decir, la posición (0, 0)) es la esquina superior izquierda, esto quiere decir que entre el borde izquierdo de la ventana y nuestro botón habrá una distancia de 60 píxeles y entre el borde superior de la ventana y el botón habrá 40 píxeles

Es posible indicar el tamaño de cualquier otro elemento de Tk utilizando los parámetros width y height, que indican el ancho y el alto en píxeles.

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

La siguiente imagen ilusta cómo influyen los cuatro argumentos (x, y, width, height) en la posición y el tamaño del widget.
Estas cuatro propiedades también pueden formularse en términos de proporción respecto del elemento padre.
Por ejemplo, podemos decirle a Tk que el tamaño del botón debe ser la mitad del tamaño de la ventana.

```
self.button.place(relwidth=0.5, relheight=0.5)
```



De este modo, cuando la ventana se expanda o se contraiga, Tk automáticamente ajustará su tamaño para que cumpla con la proporción indicada

place

Acceso al IDLE

python

Introducción a Tcl/Tk (tkinter)

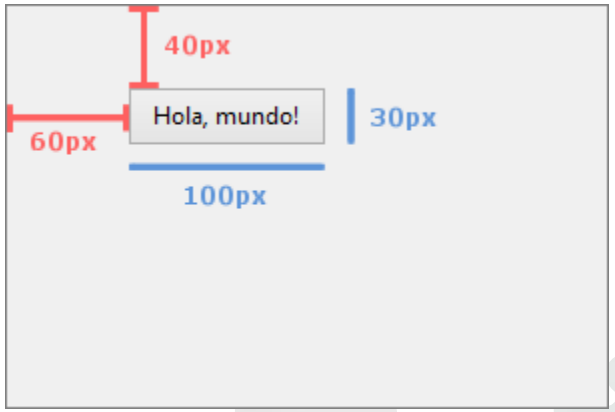
Posicionar elementos en Tcl/Tk (tkinter) **Posición absoluta (place)**

Esto explica por qué utilizamos la siguiente línea para que el marco de la aplicación (que es una instancia de ttk.Frame y que nos sirve como elemento padre) tenga siempre el mismo tamaño de la ventana.

```
self.button.place(relwidth=0.5, relheight=0.5)
```

Del mismo modo operan relx y rely, que expresan la posición de un elemento en términos proporcionales.

```
self.button.place(relx=0.1, rely=0.1, relwidth=0.5, relheight=0.5)
```



Así, al abrir el programa, cuando el tamaño de la ventana es de 300×200, el botón se encontrará en la posición (30, 20), ya que 300x0.1 = 30 y 200x0.1 = 20. A medida que el tamaño de la ventana cambie, Tk actualizará la posición del botón para que siempre cumpla con el 10% de la medida.

relwidth, relheight, relx y rely aceptan valores entre 0 y 1.

El método place() para posicionar elementos es bastante sencillo de comprender, sobre todo para usuarios de otras librerías y otros lenguajes con los que hayan desarrollado aplicaciones de escritorio. Brinda exactitud en cada uno de los objetos de nuestra interfaz y puede resultar útil en muchos casos.

El principal problema radica al momento de expandir o contraer la ventana. Si bien los argumentos proporcionales descriptos anteriormente pueden ser útiles, generalmente resultan no ser suficientes. Ubicar elementos de forma absoluta (indicando su posición como coordenadas X e Y) implica una ventana estática, que generará espacios vacíos cuando el usuario la agrande o bien se perderán algunos elementos de la vista cuando ésta se contraiga.

place

03-GUI Place.py

python

Pack: cuando añadimos un nuevo componente, se “hace hueco” a continuación de los que ya están incluidos (podemos indicar que se inserte en cualquiera de las 4 direcciones), para finalmente calcular el tamaño que necesita el widget padre para contenerlos a todos.

PYTHON pack

Posicionar elementos en Tcl/Tk (tkinter)

python

Introducción a Tcl/Tk (tkinter)

pack

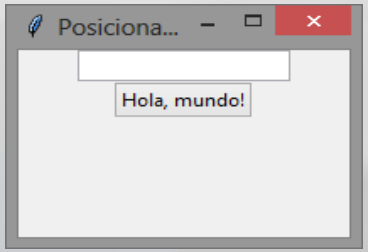
Posicionar elementos en Tcl/Tk (tkinter) **Posicionamiento relativo (pack)**

Este método es el más sencillo de los tres. En lugar de especificar las coordenadas de un elemento, simplemente le decimos que debe ir arriba, abajo, a la izquierda o a la derecha respecto de algún otro control o bien la ventana principal.

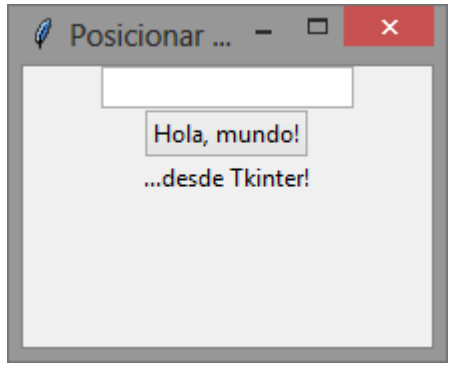
A pesar de su sencillez es muy potente y, dentro de sus limitaciones, puede resolver interfaces de usuario complejas sin perder versatilidad.

```
import tkinter as tk
from tkinter import ttk

class Application(ttk.Frame):
    def __init__(self, main_window):
        super().__init__(main_window)
        main_window.title("Posicionar elementos en Tcl/Tk")
        self.entry = ttk.Entry(self)
        self.entry.pack()
        self.button = ttk.Button(self, text="Hola, mundo!")
        self.button.pack()
        self.pack()

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

En el ejemplo, creamos una caja de texto y un botón y los ubicamos en la ventana vía la función pack. Como no indicamos ningún argumento, por defecto Tk posicionará los elementos uno arriba del otro, como se observa en la imagen.



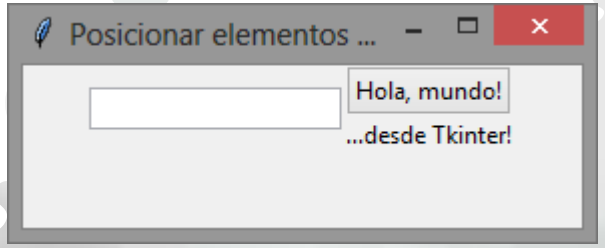
1.De modo que si añadimos otro elemento, por ejemplo, una etiqueta (ttk.Label), será ubicado debajo del botón.




```
        self.label = ttk.Label(self, text="...desde Tkinter!")
        self.label.pack()
```

La propiedad que controla la posición relativa de los elementos es side, que puede equivaler a tk.TOP (por defecto), tk.BOTTOM, tk.LEFT o tk.RIGHT. De este modo, si indicamos que la caja de texto debe ir ubicada a la izquierda, los otros dos controles se seguirán manteniendo uno arriba del otro.

```
        self.entry = ttk.Entry(self)
        self.entry.pack(side=tk.LEFT)
```



Acceso al IDLE

python

Introducción a Tcl/Tk (tkinter)

pack

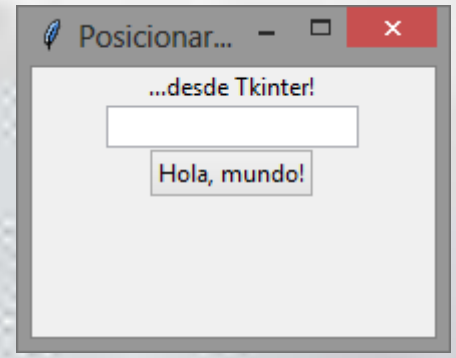
Posicionar elementos en Tcl/Tk (tkinter) **Posicionamiento relativo (pack)**

Del mismo modo, usando side=tk.RIGHT produce el efecto contrario, posicionando la caja de texto a la derecha del botón y de la etiqueta.

Te propongo que intentes por tu cuenta distintos valores para el parámetro side para comprender mejor cómo se comporta Tk.

La función pack también admite los parámetros after y before, que nos permiten controlar el orden en el que se ubican los elementos en la ventana. El siguiente ejemplo obliga a Tk a colocar la etiqueta self.label antes (before) que la caja de texto.

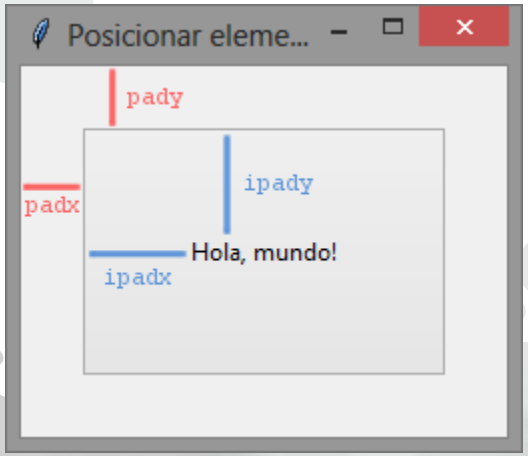
```
1.self.entry = ttk.Entry(self)
2.self.entry.pack()
3.self.button = ttk.Button(self, text="Hola, mundo!")
4.self.button.pack()
5.self.label = ttk.Label(self, text="...desde Tkinter!")
6.self.label.pack(before=self.entry)
```

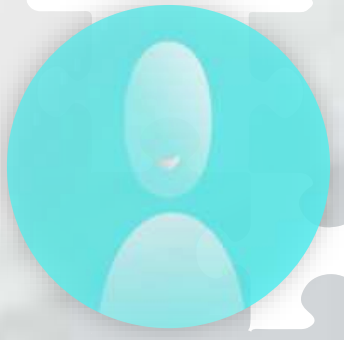


Tanto before como after aceptan como valor cualquier widget para tomar como referencia.

Otras propiedades incluyen padx, ipadx, pady y ipady que especifican (en píxeles) los margenes externos e internos de un elemento. Por ejemplo, en el siguiente código habrá un espacio de 30 píxeles entre el botón y la ventana (margen externo), pero un espacio de 50 píxeles entre el borde del botón y el texto del mismo (margen interno).

```
self.entry = ttk.Entry(self)
self.entry.pack()
self.button = ttk.Button(self, text="Hola, mundo!")
self.button.pack()
self.label = ttk.Label(self, text="...desde Tkinter!")
self.label.pack(before=self.entry)
```



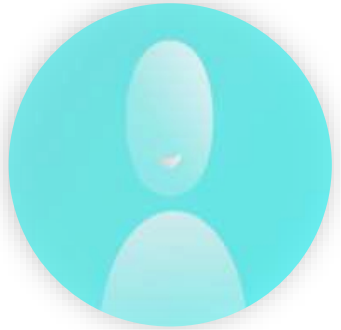


Acceso al IDLE

python

Introducción a Tcl/Tk (tkinter)

Posicionar elementos en Tcl/Tk (tkinter) **Posicionamiento relativo (pack)**

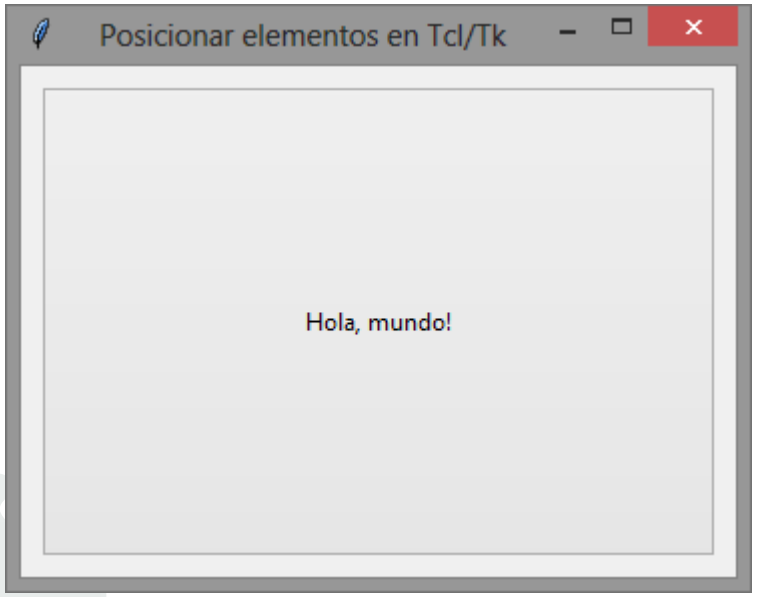


Por último, es posible especificar qué elementos deben expandirse o contraerse a medida que el tamaño de la ventana cambia, y en qué sentido deben hacerlo (vertical u horizontal), vía las propiedades expand y fill.

```
self.button = ttk.Button(self, text="Hola, mundo!")
self.button.pack(expand=True, fill=tk.X)
self.pack(expand=True, fill=tk.BOTH)
```

En el ejemplo, le indicamos al elemento padre (self) que ocupe todo el tamaño posible (expand=True) y que lo haga en ambas direcciones (fill=tk.BOTH). El botón, por otra parte, únicamente ajustará su tamaño horizontal (fill=tk.X). Si quisiéramos que se expanda solo de forma vertical, la propiedad sería fill=tk.Y o fill=tk.BOTH para expandirse en ambos sentidos.

```
self.button.pack(expand=True, fill=tk.BOTH, padx=10, pady=10)
```



Acceso al IDLE

pack

TALK IS CHEAP SHOW ME THE CODE

04-GUI Pack.py

python

Grid: la disposición de los elementos es una matriz, de manera que para cada uno debemos indicar la celda (fila y columna) que queremos que ocupe. Podemos además especificar que ocupe más de una fila y/o columna (rowspan/columnspan=3), “pegarlo” a cualquiera de los 4 bordes de la celda en vez de centrarlo (sticky=W para el borde izquierdo, por ejemplo)…

PYTHON grid

Posicionar elementos en Tcl/Tk (tkinter)

python

Introducción a Tcl/Tk (tkinter)

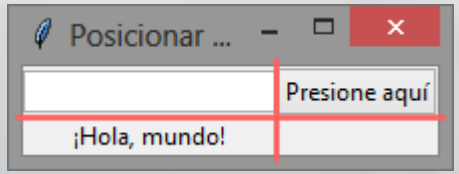
grid

Posicionar elementos en Tcl/Tk (tkinter) **Manejo en forma de grilla (grid)**

El método de grilla siempre es una buena elección, desde pequeñas hasta grandes y complejas interfaces. Consiste en dividir conceptualmente la ventana principal en filas (rows) y columnas (columns), formando celdas en donde se ubican los elementos. Veamos un ejemplo.

```
import tkinter as tk
from tkinter import ttk

class Application(ttk.Frame):
    def __init__(self, main_window):
        super().__init__(main_window)
        main_window.title("Posicionar elementos en Tcl/Tk")
        main_window.columnconfigure(0, weight=1)
        main_window.rowconfigure(0, weight=1)
        self.entry = ttk.Entry(self)
        self.entry.grid(row=0, column=0)
        self.button = ttk.Button(self, text="Presione aquí")
        self.button.grid(row=0, column=1)
        self.label = ttk.Label(self, text="¡Hola, mundo!")
        self.label.grid(row=1, column=0)
        self.grid(sticky="nsew")

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

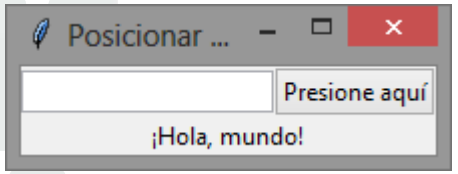
Por el momento vamos a concentrarnos únicamente en el código entre las líneas 9 y 16. Allí se crean tres controles (una caja de texto, un botón y una etiqueta) y vía la función grid se especifica su posición en la grilla.



Como se observa en la imagen, nuestra grilla consta hasta el momento de dos filas y dos columnas, dando un total de (dos por dos es cuatro, ¿no?) cuatro celdas.

La caja de texto está en la columna 0 y la fila 0. Siguiendo esta convención, el botón está en la celda (1, 0) y la etiqueta en la posición (0, 1). La celda (1, 1) no contiene ningún elemento. Una grilla puede tener tantas columnas y filas como queramos.
Podemos indicarle a un elemento que debe ocupar más de una fila o columna. Por ejemplo, ya que la celda (1, 1) está vacía, nuestra etiqueta podría ocuparla para que el diseño sea más agradable.

```
self.label.grid(row=1, column=0, columnspan=2)
```



columnspan indica cuántas columnas debe ocupar el control (por defecto 1). El parámetro rowspan opera de forma similar para las filas.

Acceso al IDLE

python

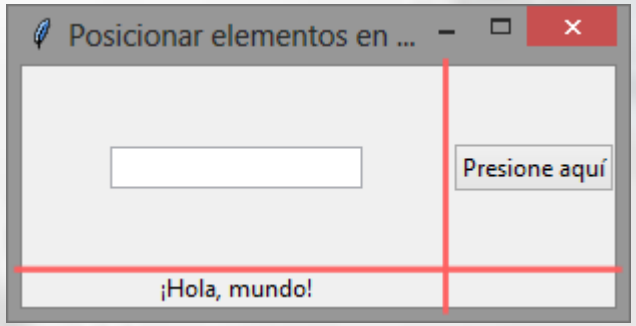
Introducción a Tcl/Tk (tkinter)

Posicionar elementos en Tcl/Tk (tkinter)

Manejo en forma de grilla (grid)

Por defecto las columnas y las filas no se expanden o contraen si la ventana cambia su tamaño. Para esto, usamos las funciones rowconfigure y columnconfigure con el parámetro weight. Por ejemplo, el siguiente código indica que la columna 0 y la fila 0 deben expandirse.

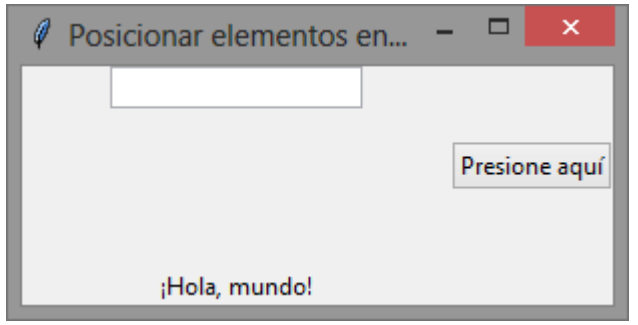
```
        # Expandir horizontalmente a columna 0.
        self.columnconfigure(0, weight=1)
        # Expandir verticalmente la fila 0.
        self.rowconfigure(0, weight=1)
```

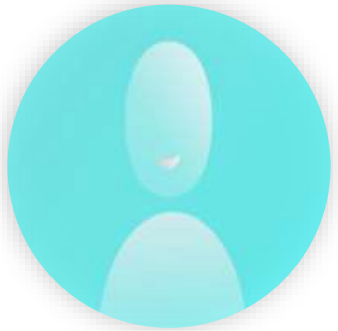


La imagen muestra cómo se ha expandido la celda una vez agrandada la ventana y cómo nuestra caja de texto se mantuvo en el centro. Para que el elemento se posicione arriba, abajo, a la derecha o izquierda de la celda que lo contiene, podemos usar el parámetro sticky con las opciones "n" (norte), "s" (sur), "e" (este) o "w" (oeste), respectivamente

```
# Anclar en la parte superior (n) de la celda.
        self.entry.grid(row=0, column=0, sticky="n")
```

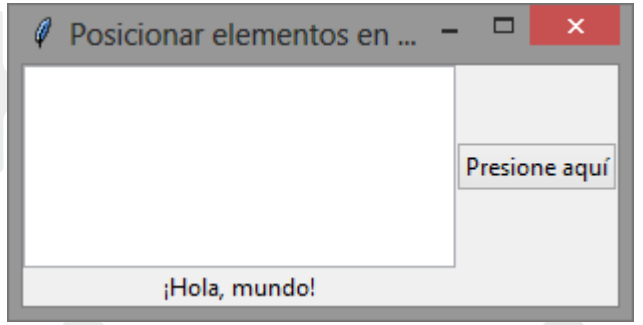
grid





Combinando estas propiedades, podemos lograr que el widget se expanda de forma horizontal ("ew"), vertical ("ns") o en ambas direcciones ("nsew").

```
        # Expandir en todas las direcciones.
        self.entry.grid(row=0, column=0, sticky="nsew")
```



La función grid acepta, al igual que pack(), los argumentos padx, pady, ipadx, ipady para establecer márgenes.

```
self.entry.grid(row=0, column=0, sticky="nsew", padx=10, pady=10)
```

Acceso al IDLE

python

Introducción a Tcl/Tk (tkinter)

Posicionar elementos en Tcl/Tk (tkinter)

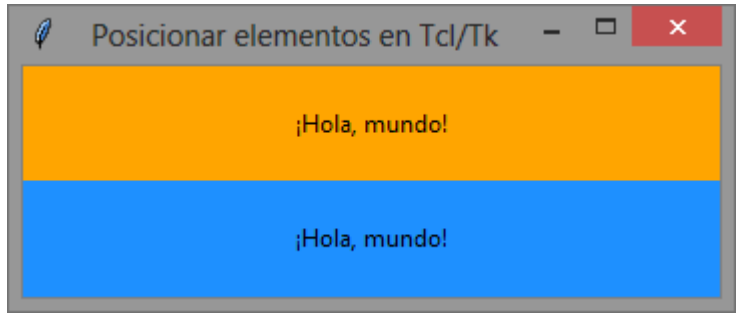
Manejo en forma de grilla (grid)

Por último, el método de grillas en Tk permite configurar en qué medida se expanden las columnas y filas. Por ejemplo, consideremos el siguiente código.

```
        def __init__(self, main_window):
            super().__init__(main_window)
            main_window.title("Posicionar elementos en Tcl/Tk")
            main_window.columnconfigure(0, weight=1)
            main_window.rowconfigure(0, weight=1)
            self.label1 = tk.Label(
            self, text="¡Hola, mundo!", bg="#FFA500")
            self.label1.grid(row=0, column=0, sticky="nsew")
            self.label2 = tk.Label(
            self, text="¡Hola, mundo!", bg="#1E90FF")
            self.label2.grid(row=1, column=0, sticky="nsew")
            self.grid(sticky="nsew")
            self.columnconfigure(0, weight=1)
            self.rowconfigure(0, weight=1)
            self.rowconfigure(1, weight=1)
```

grid

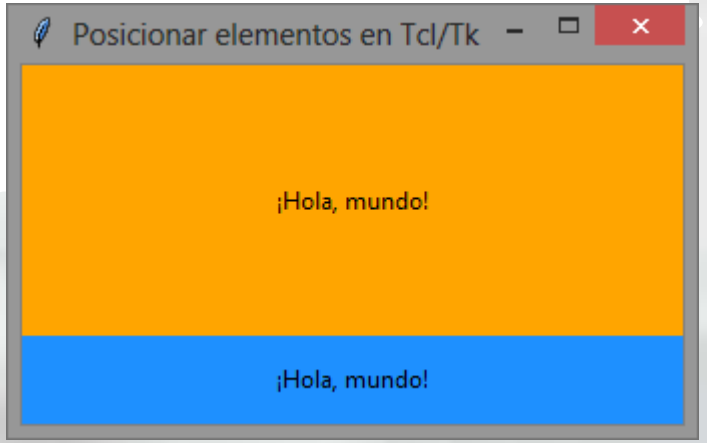
En esta ventana creamos dos etiquetas y las ubicamos en la misma columna (0) pero en diferentes filas (0 y 1). Luego, vía rowconfigure y columnconfigure indicamos que deben expandirse y contraerse junto con la ventana.



La imagen nos muestra cómo los dos elementos comparten el espacio disponible, de modo que uno siempre tiene el mismo tamaño que el otro, independientemente de cuán chica o grande sea la ventana. Pero en ocasiones es deseable que un control se expanda más que otro y viceversa. Para esto, podemos aumentar el «peso» (weight) que ejerce una fila o columna a Tk al momento de expandirse.

```
self.rowconfigure(0, weight=5)
self.rowconfigure(1, weight=1)
```

Con esta configuración, la fila 0 (correspondiente a la etiqueta naranja) siempre tendrá un tamaño cinco veces mayor a la fila 1, como observamos a continuación.



Acceso al IDLE

python

Introducción a Tcl/Tk (tkinter)

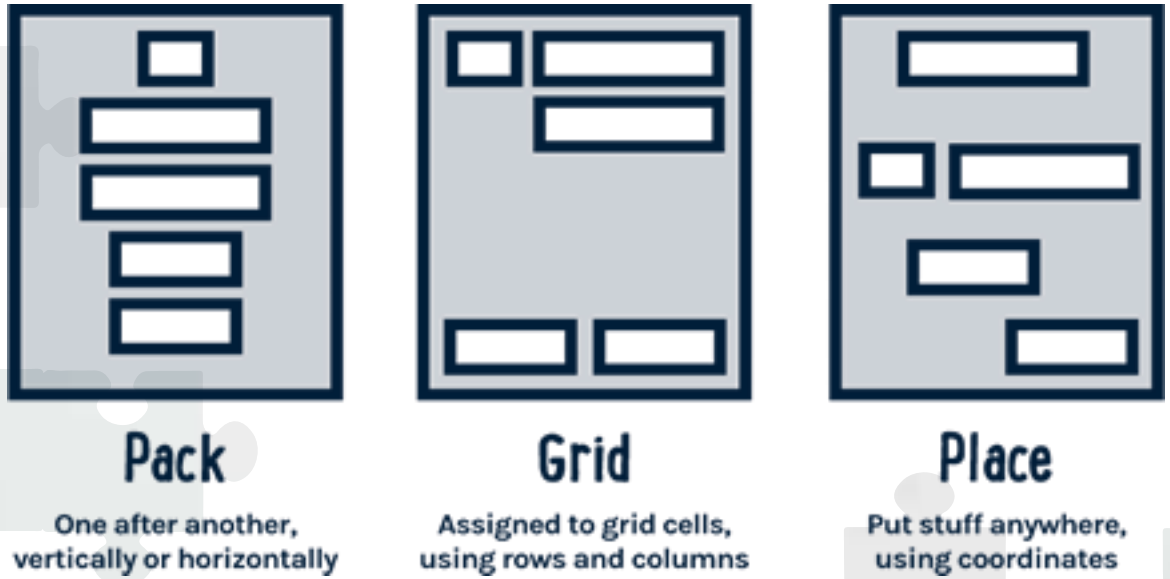
Posicionar elementos en Tcl/Tk (tkinter)
Conclusión

Place – Pack - grid

El método place() brinda total control sobre la ubicación de cada uno de los elementos, pues su posición es absoluta. Esto es generalmente conveniente para pequeñas y medianas interfaces que se mantegan estáticas y no tengan aspiración de ser expandidas.

pack(), por su parte, es bastante sencillo de utilizar y con él pueden obtenerse interfaces ricas y complejas. Sin embargo, el hecho de que la posición de cada widget dependa de otro puede generar complicaciones al momento de realizar cambios, sobre todo modificaciones a aplicaciones ya existentes.

or último, el manejo a través de una grilla, como comentaba anteriormente, es siempre una buena elección. Conociendo todas sus propiedades y aplicándolas cuidadosamente podremos obtener desde pequeñas hasta grandes interfaces de usuario completamente adaptables y fáciles de utilizar.

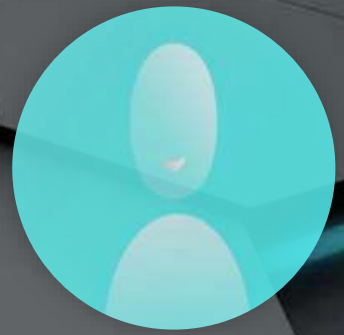


04-GUI Grid.py

PYTHON

DataBase Sqlite

¿Se puede usar SQLite con Python? SQLite es una base de datos SQL autónoma basada en archivos. SQLite viene incluido con Python y se puede usar en cualquiera de sus aplicaciones Python sin tener que instalar ningún software adicional.



SQLite en PYTHON.

SQLite

DataBase Sqlite

SQLite es un sistema de gestión de base de datos relacional ligero y autónomo, que se integra directamente dentro de una aplicación, sin necesidad de un servidor separado. Es ideal para aplicaciones con requisitos de datos modestos, como aplicaciones móviles, sistemas embebidos y proyectos pequeños.

SQLite en PYTHON.

SQLite : Base de datos desde Python

SQLite también es un gestor de bases de datos relacional pero con objetivos muy diferentes a MySQL, SQLServer, Oracle etc.

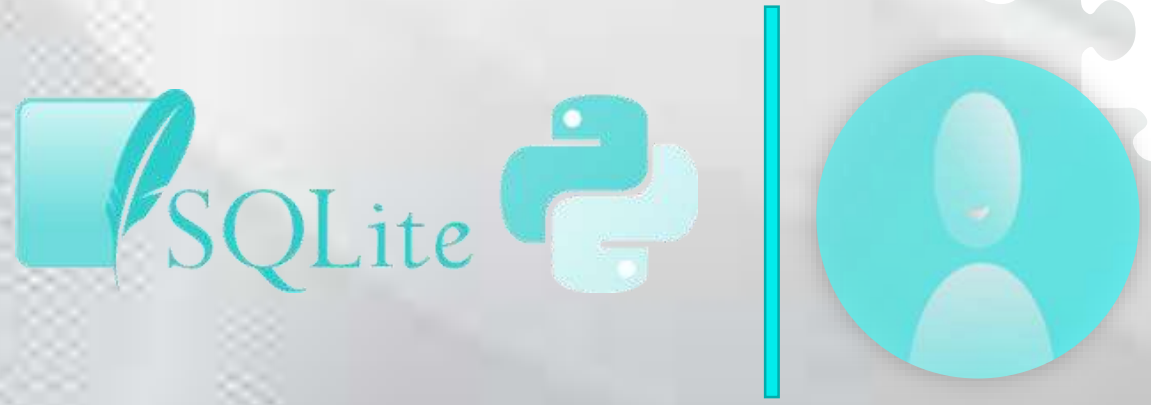
Este gestor de base de datos tiene por objetivo ser parte de la misma aplicación con la que colabora, es decir no cumple los conceptos de cliente y servidor.

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..

```
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:

python

SQLite : Base de datos desde Python

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()
```

Acceso al IDLE

01 CreateTable.py

python

SQLite : Base de datos desde Python

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.

```
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()

Acceso al IDLE

02 Insertar.py

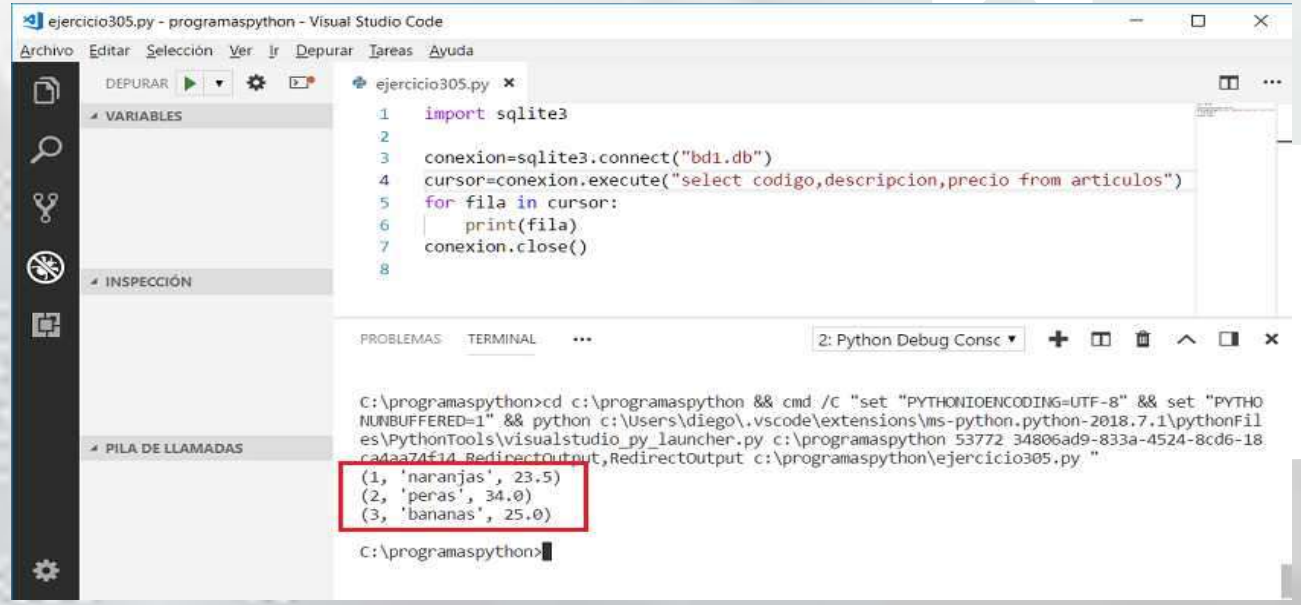
SQLite : Base de datos desde Python

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.

```
import sqlite3

conexion=sqlite3.connect("bd1.db")
cursor=conexion.execute("select codigo,descripcion,precio from
articulos")
for fila in cursor:
    print(fila)
conexion.close()
```

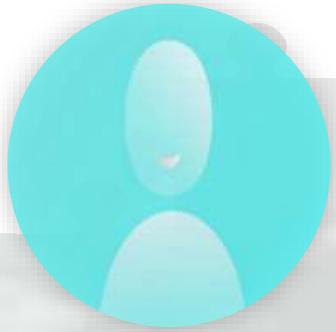
Si ejecutamos este programa luego de haber cargado las tres filas del ejercicio anterior el resultado será el siguiente:



El método execute retorna un objeto de la clase Cursor:

cursor=conexion.execute("select codigo,descripcion,precio from articulos")

Acceso al IDLE



SQLite : Base de datos desde Python

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.

```
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()
```

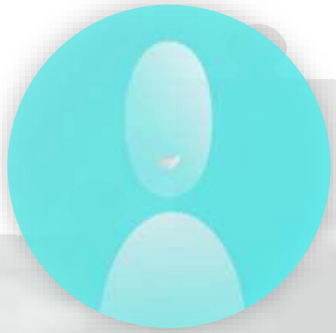
El resultado de este comando SQL select puede ser de una fila si existe el código de artículo ingresado o cero filas:

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.")
```

Acceso al IDLE



SQLite : Base de datos desde Python

Actualizar una fila de una tabla.

Implementaremos un programa que actualice unl valor

```
import sqlite3

conexion=sqlite3.connect("bd1.db")
conexion.execute("update articulos set descripcion = 'palta' where
codigo=1 ")
conexion.commit()
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()

Acceso al IDLE

SQLite : Base de datos desde Python

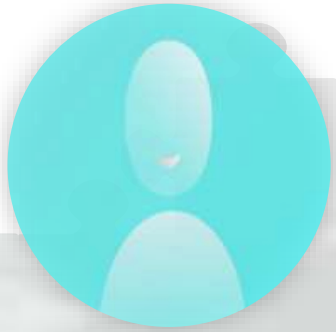
Borrar una fila de una tabla.

Implementaremos un programa que actualice unl valor

```
import sqlite3

conexion=sqlite3.connect("bd1.db")
conexion.execute("delete from articulos where codigo=1 ")
conexion.commit()
conexion.close()
```

El resultado de este comando SQL select puede ser de una fila si existe el código de artículo ingresado o cero filas:

Acceso al IDLE



TALK IS CHEAP SHOW ME THE CODE

06 Borrar.py

python

PYTHON

GUI & DataBase

PYTHON & SQLite & TKinter

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.

Python GUI & DataBase SQLite

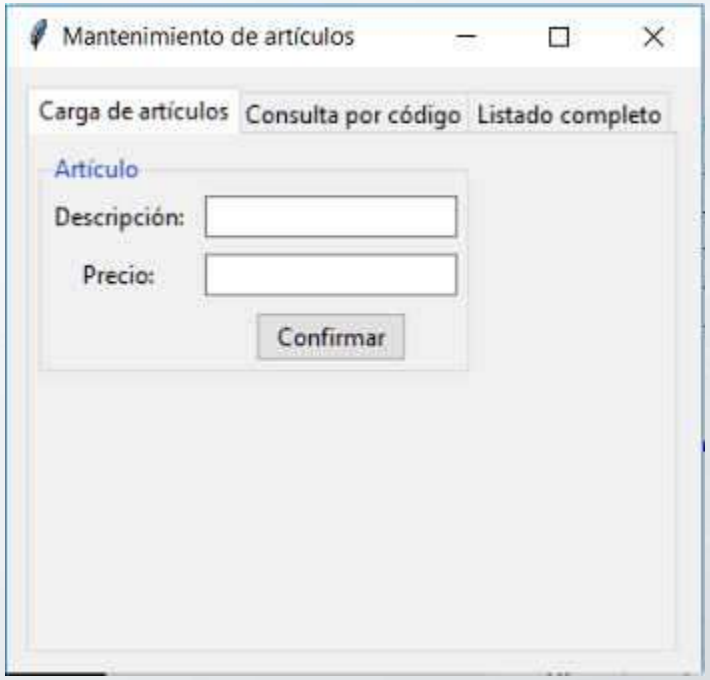
Es muy posible que necesitemos en algunas situaciones acceder a una base de datos de SQLite desde una aplicación con una interfaz visual.
Implementaremos el mismo problema que resolvimos cuando trabajamos con el gestor de base de datos MySQL.

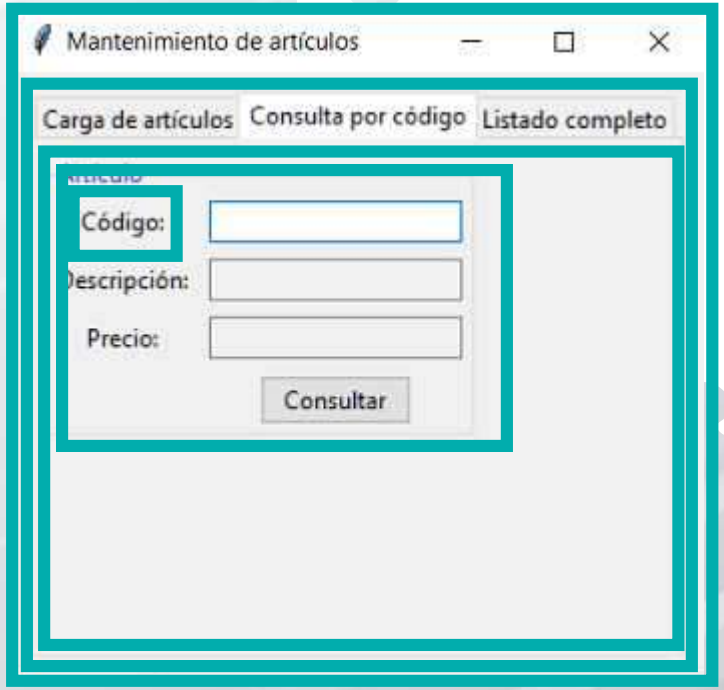
Problema:

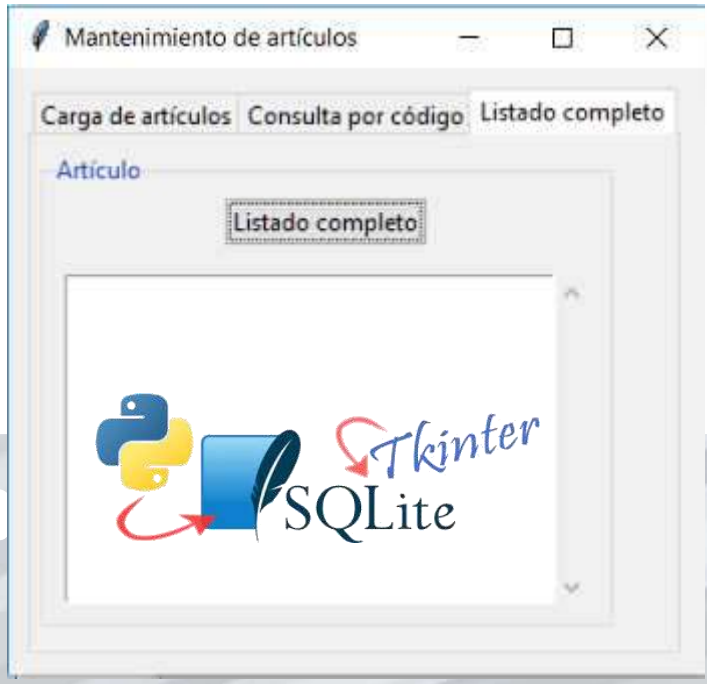
Desarrollar una aplicación visual con la librería tkinter que permita implementar los algoritmos de carga de artículos, consulta por código y listado completo.

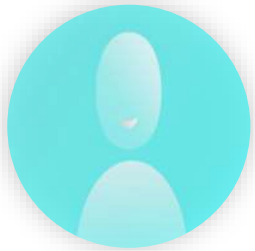
Trabajaremos con la base de datos 'bd1.db' que creamos en el concepto anterior.
Las interfaz visual para la carga debe ser:

```
self.articulo1=articulos.Articulos()
self.ventana1=tk.Tk()
self.ventana1.title("Mantenimiento de artículos")
self.cuaderno1 = ttk.Notebook(self.ventana1)

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:")
```





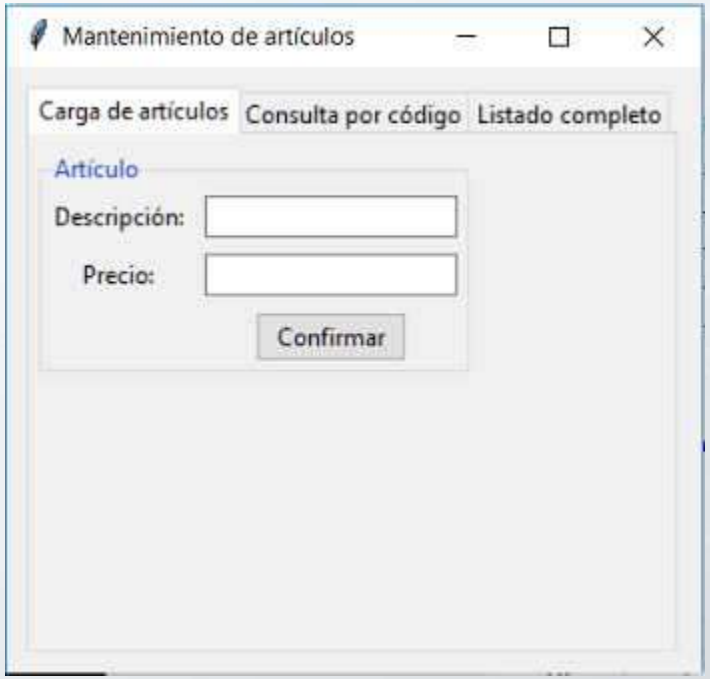


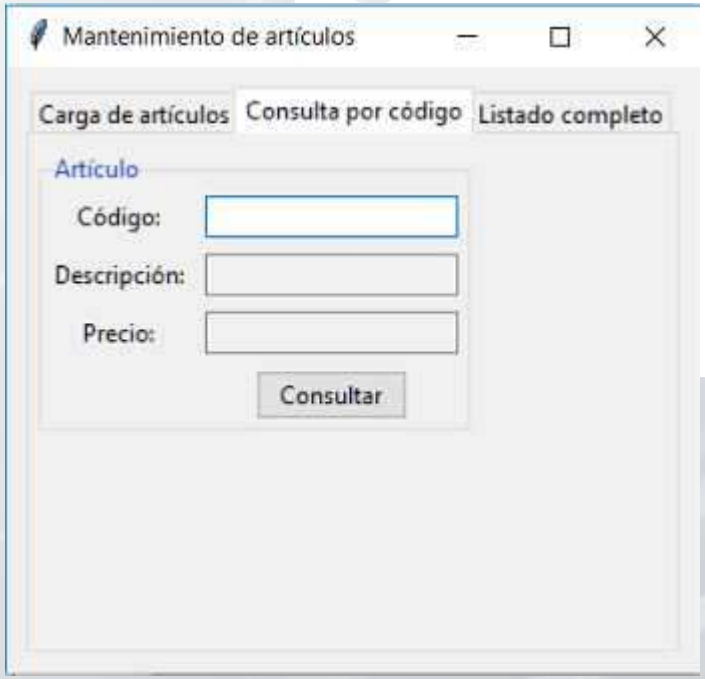


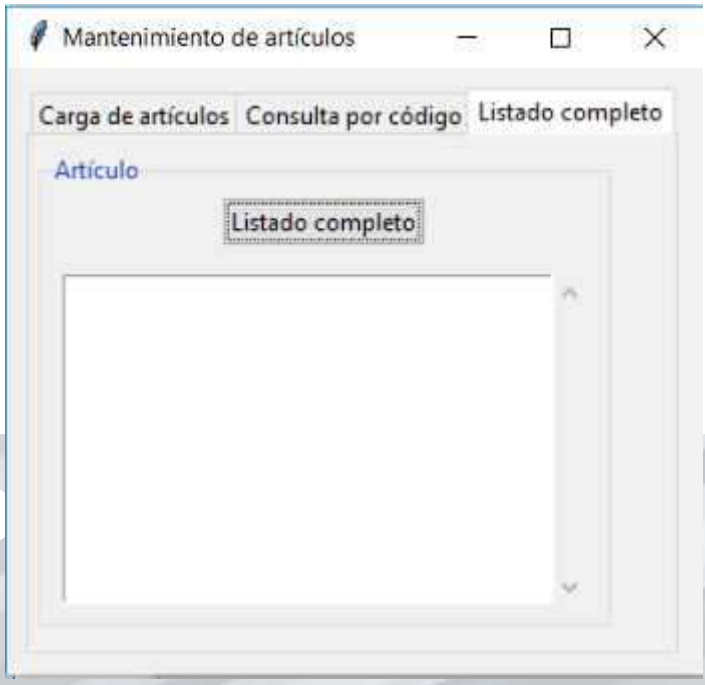
python

Python GUI & DataBase SQLite

Para trabajar un poco más ordenado en la resolución de este problema lo dividiremos en dos módulos **'formularioarticulos.py' y 'articulos.py'.**







https://www.tutorialesprogramacionya.com/pythonya/detalleconcepto.php?punto=88&codigo=89&inicio=75

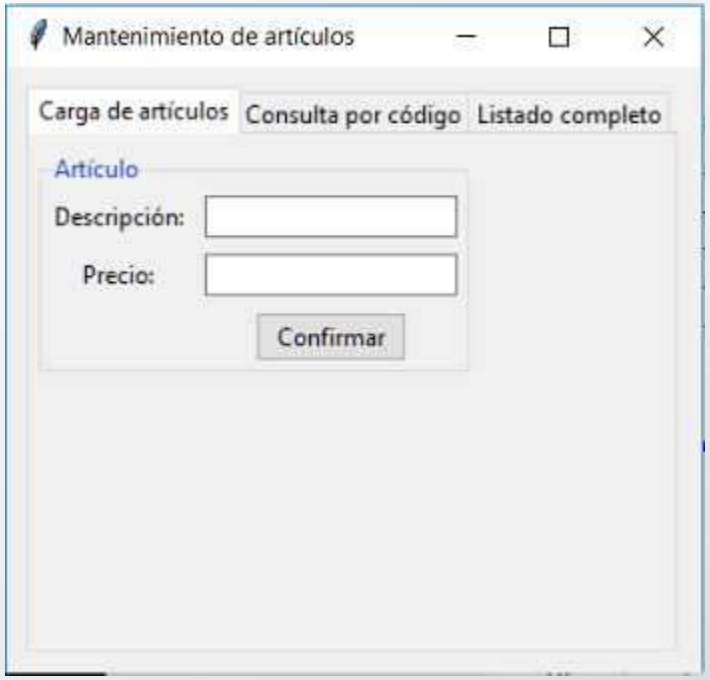
Python GUI & DataBase SQLite

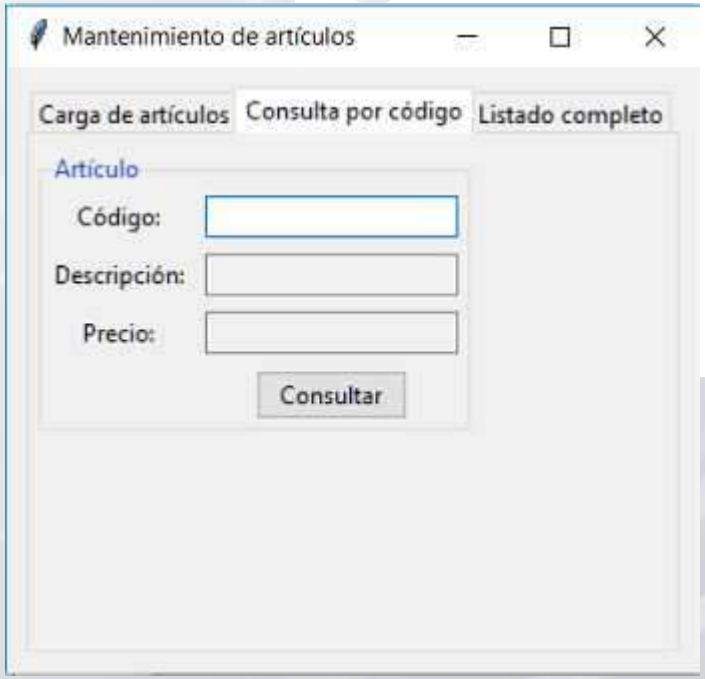
Es muy posible que necesitemos en algunas situaciones acceder a una base de datos de SQLite desde una aplicación con una interfaz visual.

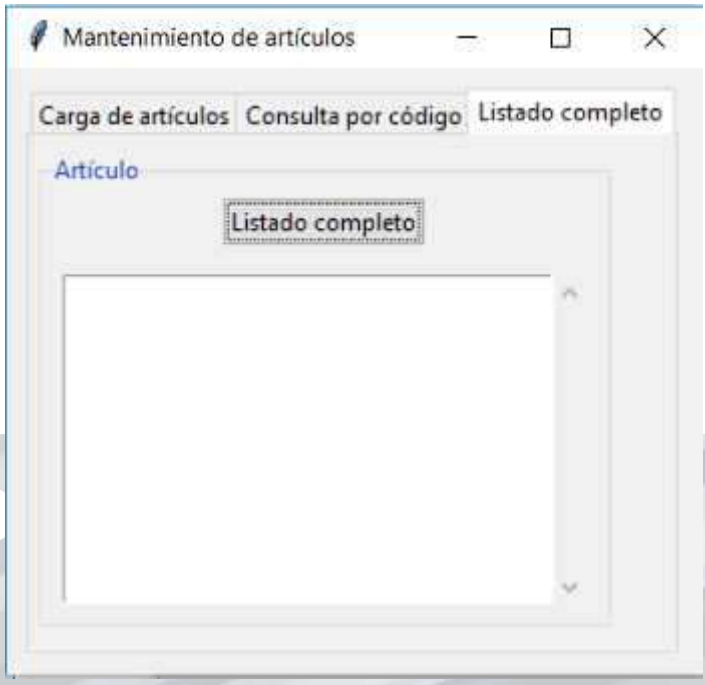
Problema:

Desarrollar una aplicación visual con la librería tkinter que permita implementar los algoritmos de carga de artículos, consulta por código y listado completo.

Trabajaremos con la base de datos 'bd1.db' que creamos en el concepto anterior.
Las interfaz visual para la carga debe ser:







Python GUI & DataBase SQLite

Tener en cuenta que el módulo principal se encuentra en el archivo 'formularioarticulos.py' y es el que debemos ejecutar:

```
import articulos
```

Cuando al programa lo ejecutemos desde la línea de comandos fuera del editor VS Code debemos recordar de llamar al módulo principal:

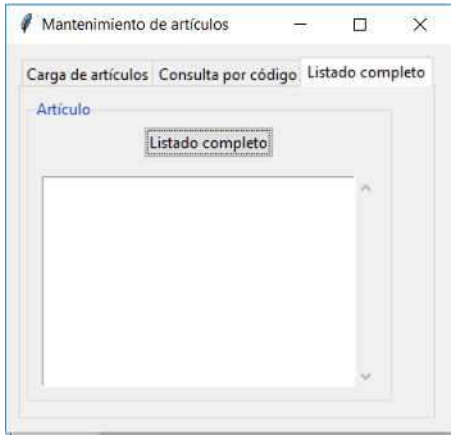
modulos python sqlite

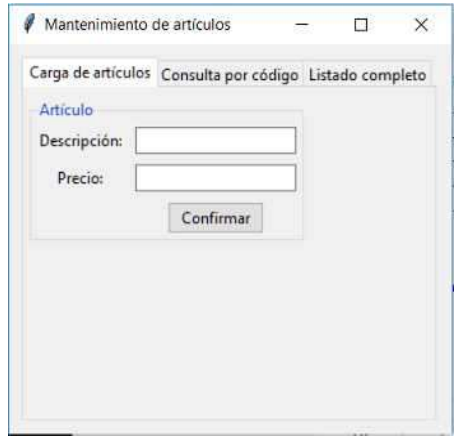
Analicemos un poco el código del módulo 'formularioarticulos.py', lo primero que hacemos es importar los módulos necesarios para implementar la interfaz visual:

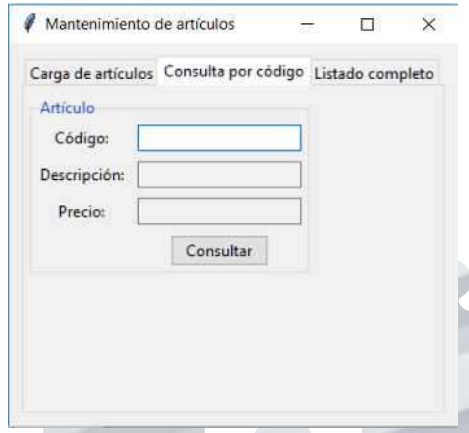
```
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox as mb
from tkinter import scrolledtext as st
import articulos
```

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
```







python

Python GUI & DataBase SQLite

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:

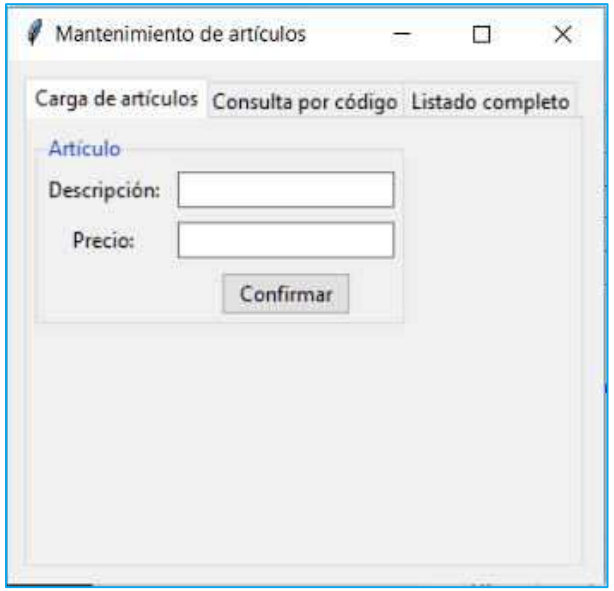
class FormularioArticulos:

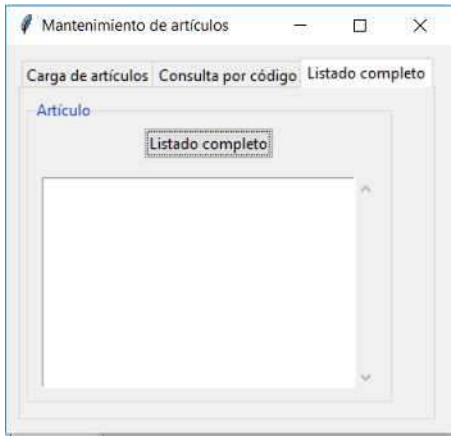
```
class FormularioArticulos:
    def __init__(self):
        self.articulo1=articulos.Articulos()
```

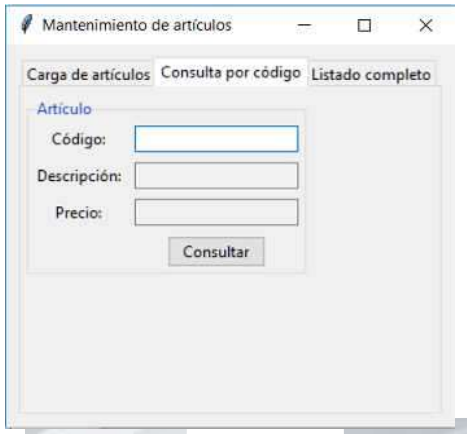
También en el método __init__ llamamos a una serie de métodos para crear cada una de las páginas del objeto de la clase 'Notebook':

```
    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()
```

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())
        self.articulo1.alta(datos)
        mb.showinfo("Información", "Los datos fueron cargados")
        self.descripcioncarga.set("")
        self.preciocarga.set("")
```

python

Python GUI & DataBase SQLite

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:

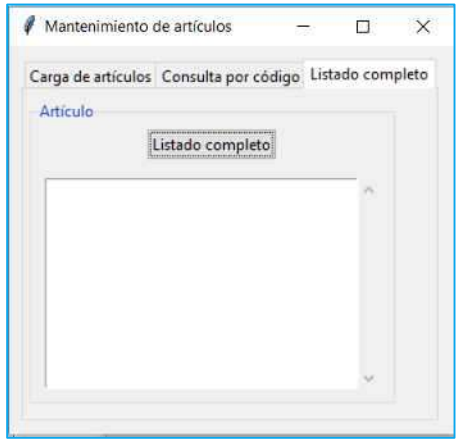
```
class FormularioArticulos:
    def __init__(self):
        self.articulo1=articulos.Articulos()
```

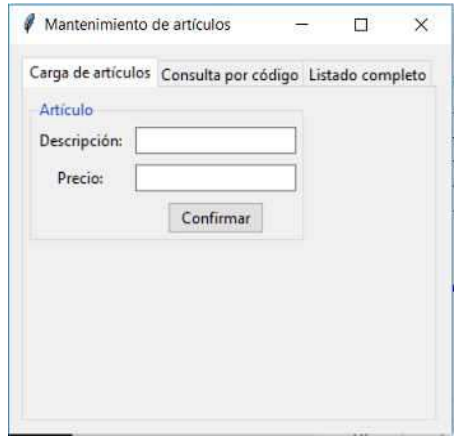
También en el método __init__ llamamos a una serie de métodos para crear cada una de las páginas del objeto de la clase 'Notebook':

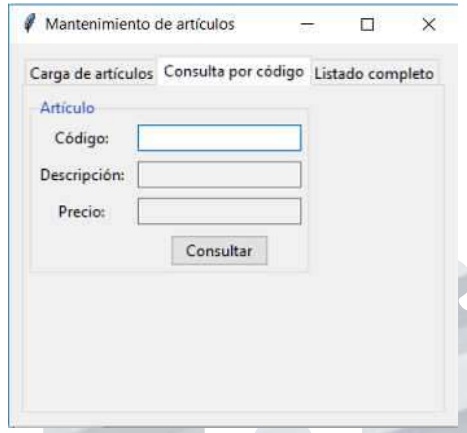
```
        self.ventana1=tk.Tk()
        self.ventana1.title("Mantenimiento de artículos")
        self.cuaderno1 = ttk.Notebook(self.ventana1)
        self.carga_articulos()
        self.consulta_por_codigo()
        self.listado_completo()
        self.borrado()
        self.modificar()
        self.cuaderno1.grid(column=0, 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())
```







python

Python GUI & DataBase SQLite

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")
```

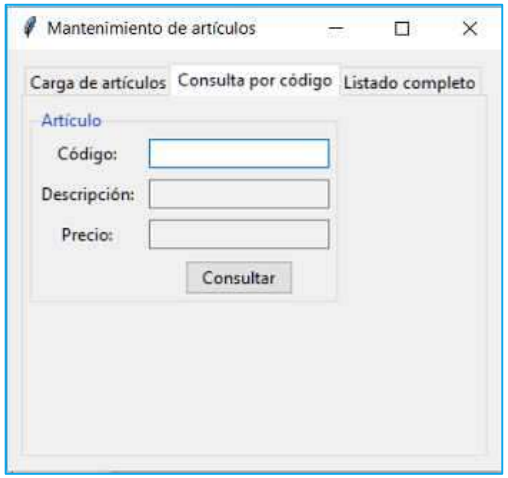
Creamos una tubla con un solo dato (es obligatoria la coma para que Python lo considere una tupla:

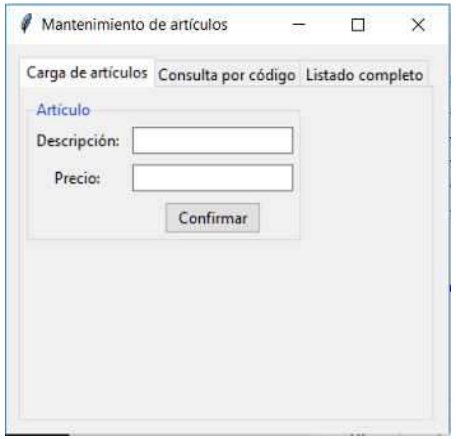
```
    datos=(self.codigo.get(), )
```

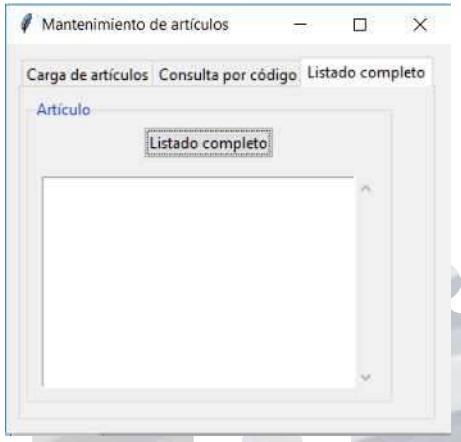
Llamamos al método consulta de la clase 'Articulos' que se encuentra en el otro módulo. El método 'consulta' retorna una lista vacía si no existe el código de artículo ingresado o una lista con una tupla en su interior.

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()
```







python

Python GUI & DataBase SQLite

Listado completo

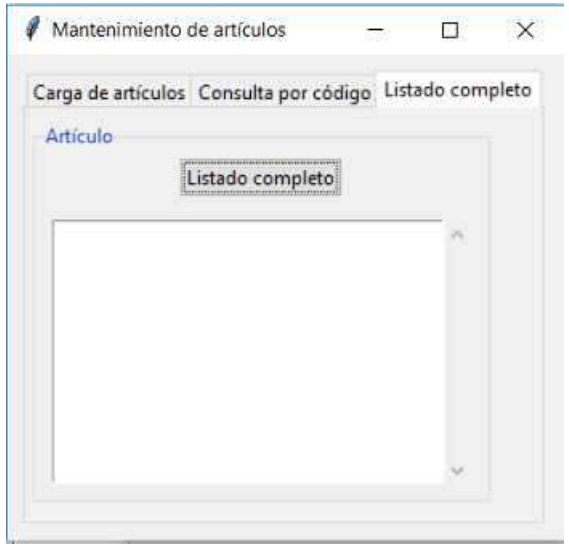
Para mostrar todas las filas de la tabla 'articulos' hemos dispuesto un objeto de la clase 'scrolledtext':

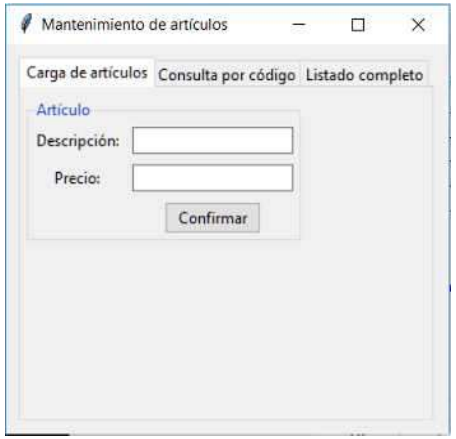
```
    def listar(self):
        respuesta=self.articulo1.recuperar_todos()
        self.scrolledtext1.delete("1.0", tk.END)
        for fila in respuesta:
            self.scrolledtext1.insert(tk.END, "código:"+str(fila[0])+
                                              "\ndescripción:"+fila[1]+
                                              "\nprecio:"+str(fila[2])+"\n\n"
)
```

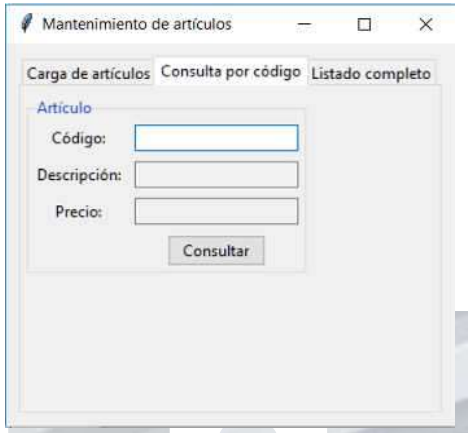
Llamamos al método 'recuperar_todos' de la clase 'Articulos' y obtenemos una lista con un conjunto de tuplas con cada fila de la tabla.

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()
```

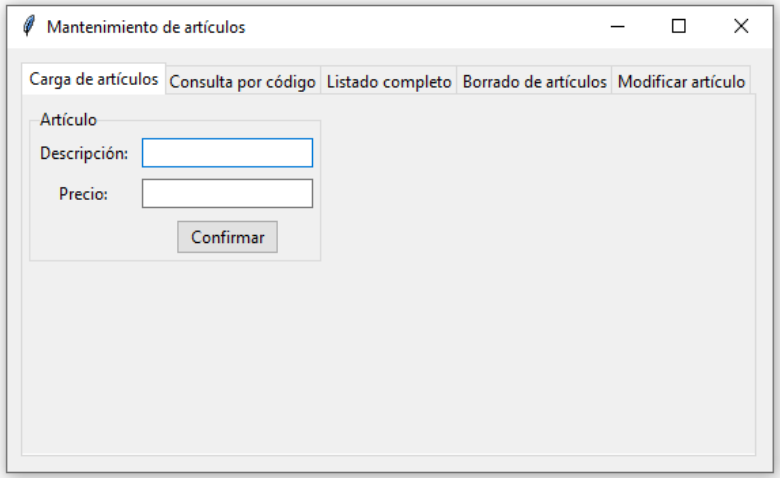


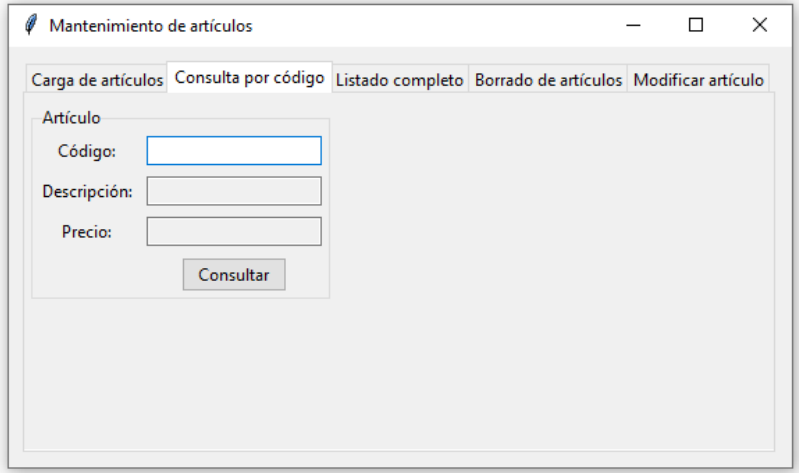


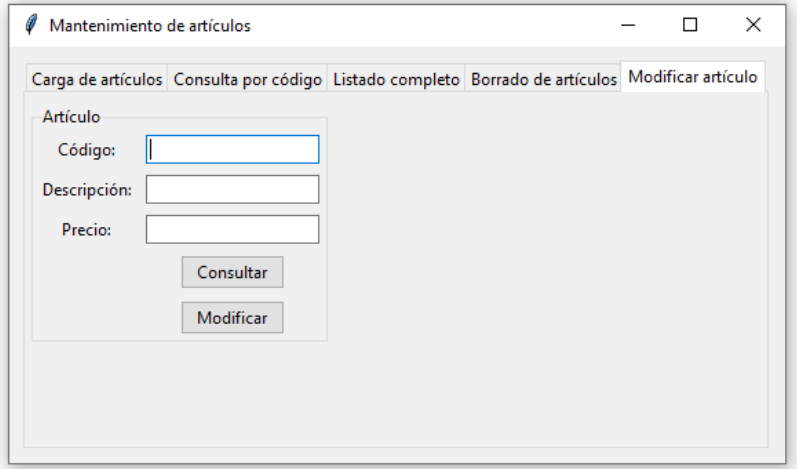


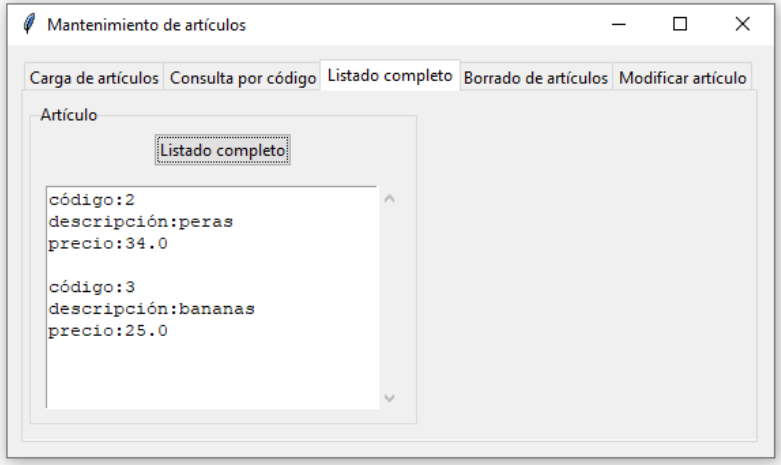
python

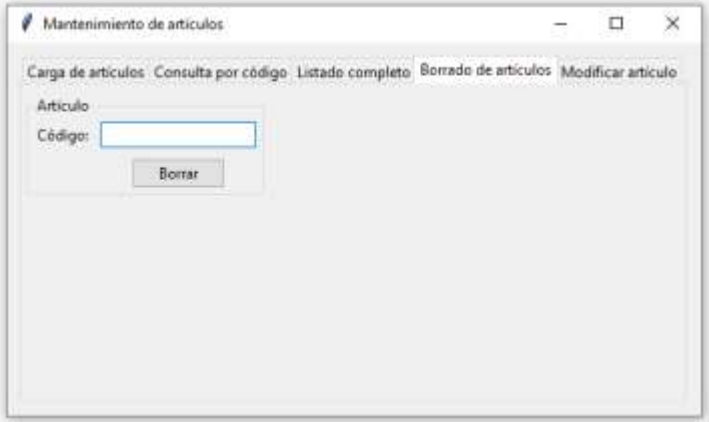
Python GUI & DataBase SQLite

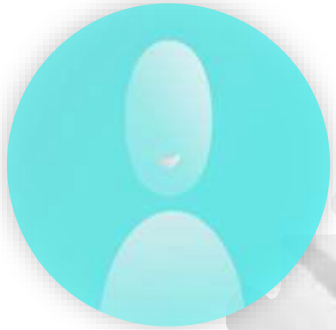












formularioarticulos.py

articulos.py

python

Muchas Gracias.