Herramientas / frameworks

## PyQt / PySide

PyQt y PySide son “bindings” de Python para el framework Qt, que está escrito en C++. Qt es un toolkit muy completo para desarrollar aplicaciones con interfaces gráficas, con soporte para ventanas, widgets (botones, cajas de texto, tablas, vistas, etc.), gráficos 2D/3D, multimedia, redes, hilos, entre muchas otras funcionalidades. Las versiones más recientes están diseñadas para integrarse bien con Python moderno. PySide a veces es llamado “Qt for Python”.

Ambos te permiten construir con Python interfaces completas que lucen y se comportan bien en múltiples plataformas (Windows, macOS, Linux). Al usar Qt, tienes acceso a la infraestructura madura de ese ecosistema.

Usos frecuentes:

* Herramientas visuales de diseño: Qt Designer (o Qt Creator) permite diseñar visualmente las ventanas y formularios, y guardarlos como archivos .ui. Luego estos archivos se pueden cargar o convertir a código Python automáticamente. Esto agiliza mucho la parte de diseño visual.
* Arquitectura robusta: Qt tiene un sistema de señales y slots bastante estructurado, lo que permite separar la parte visual de la lógica de negocios de forma elegante.
* Amplio ecosistema: Con Qt tienes acceso a muchos módulos (redes, gráficos, animaciones, acceso a hardware, OpenGL, etc.), lo cual permite hacer aplicaciones complejas sin necesidad de añadir muchas dependencias externas
* Versiones modernas: PySide6 / Qt6 permiten aprovechar tecnologías más recientes y mantener compatibilidad con versiones actuales de Python.
* Compatibilidad multiplataforma: Al basarse en Qt, permite que tu aplicación corra con mínimas modificaciones en distintos sistemas operativos.
* Curva de aprendizaje: Aunque potente, tiene muchos conceptos (layouts, vistas de modelo, señales/slots, jerarquía de widgets) que requieren tiempo para dominar bien.
* Empaquetado: Las aplicaciones grandes pueden volverse pesadas al incluir todos los componentes de Qt, y empaquetarlas para distribución puede requerir cuidado (dependencias, rutas, librerías compartidas).
* Consistencia visual: Qt permite aplicar temas, estilos, soporte para visualización nativa o personalizada, lo que dota de flexibilidad a la apariencia de la interfaz.

Kivy  
Es un framework de Python orientado a la creación de aplicaciones con interfaces gráficas que pueden ejecutarse tanto en escritorio como en dispositivos móviles (Android, iOS). Se enfoca en soportar **entrada táctil / gestos / multitouch** y tiene un motor gráfico propio basado en OpenGL / GPU para renderizado acelerado.

Se distingue porque no se preocupa tanto por imitar los controles nativos del sistema, sino por ofrecer una interfaz coherente, moderna y flexible, incluso en dispositivos táctiles.

Usos frecuentes

* Diseño independiente de plataforma: puedes usar el mismo código base para escritorio y móvil (aunque con ajustes).
* Lenguaje KV: Kivy ofrece un lenguaje de diseño (KV) que separa la definición visual de la lógica, lo que facilita el mantenimiento del código.
* Animaciones, transiciones, gestos: Kivy está pensado para interfaces dinámicas y modernas, donde puedas tener arrastres, “swipes”, efectos visuales, etc.
* No apariencia nativa: Los widgets de Kivy tienen su propio estilo visual, no se ven como los controles del sistema operativo (lo que puede hacer que la app se sienta distinta).
* Rendimiento: al usar OpenGL, puede aprovechar aceleración por hardware, pero si la interfaz es muy compleja puede necesitar optimización.
* Flexibilidad en el diseño: no estás tan atado a las restricciones de los controles nativos; puedes crear interfaces más libres.
* Curva de dominio: entender bien el sistema de layouts, propiedades, binding, administración de eventos en Kivy puede llevar tiempo.
* Ecosistema móvil: aunque soporta desplegar en Android / iOS, no siempre es trivial empaquetar o lidiar con permisos y particularidades de cada plataforma.

## wxPython (wxWidgets + Python)

wxPython es un binding de Python para la biblioteca C++ wxWidgets. wxWidgets es un toolkit multiplataforma que emplea los controles nativos del sistema para que las aplicaciones “se vean como del sistema”. wxPython envuelve esos controles para que puedan usarse desde Python.

Usos frecuentes

* Apariencia nativa: dado que usa controles reales del sistema cuando es posible, la interfaz tiende a integrarse visualmente con el entorno del usuario.
* Madurez: es una herramienta veterana, con muchas versiones y pruebas en producción
* Gran set de widgets: wxPython incluye muchos controles (listas, rejillas, árboles, pestañas, paneles, etc.).
* Gestión de eventos: usa un modelo de vínculo (bind) para eventos; puedes ligar funciones o métodos a eventos específicos del widget.
* Layouts y “sizers”: wxPython utiliza sizers para administrar disposiciones dinámicas (ajustar tamaños, redistribuir controles cuando la ventana cambia).
* Complejidad moderada: aunque no es tan pesada como Qt en algunos aspectos, manejar diseño flexible, adaptación a distintos tamaños de ventana, etc., puede volverse trabajoso.
* Empaquetado multiplataforma: debe cuidarse al distribuir la aplicación para que las dependencias con librerías nativas estén correctamente incluidas.
* Menor ecosistema gráfico/multimedia: en comparación con Qt, algunas características más avanzadas (gráficos 3D, módulos multimedia) pueden requerir dependencias adicionales externas.

## PyGUI

Es un proyecto con la intención de ofrecer una API de GUI “pythonica” — es decir, más cercana al estilo y filosofía de Python. Su objetivo es ser ligera, fácil de usar, con una capa mínima entre el desarrollador y los componentes del sistema donde sea posible.

Usos frecuentes

* Ligera: pretende no imponer demasiada infraestructura, mantenerse cercana al desarrollador.
* Controles nativos: donde sea posible, puede usar componentes nativos para que la interfaz no luzca “ajena” al sistema.
* API simple: menos abstracciones complejas, con la intención de que escribir interfaces básicas sea directo.
* Menos funciones avanzadas: no ofrece tantas características “listas para usar” como un framework grande (por ejemplo, gráficos complejos, manejo de multimedia, animaciones complejas).
* Menor comunidad / soporte: al no ser tan usado, puede tener menos documentación, ejemplos, soporte de terceros.
* Riesgo de obsolescencia: proyectos más pequeños pueden no actualizarse con la misma frecuencia, con lo cual pueden quedarse rezagados ante cambios en versiones de Python o de sistemas operativos.

|  |  |  |
| --- | --- | --- |
| **Herramientas** | **ventajas** | **Desventajas** |
| **PyQt / PySide (Qt)** | Muy completo, gran variedad de widgets, herramientas visuales (Qt Designer), soporte multiplataforma robusto | Curva de aprendizaje relativamente alta; tamaño del framework; licencias pueden ser restrictivas |
| **Kivy** | Permite apps móviles + desktop, soporte multitouch, diseño flexible | Apariencia no nativa, rendimiento puede requerir optimización, curva moderada |
| **wxPhyton** | Componentes nativos que lucen integrados, estabilidad y madurez | Ecosistema más limitado que Qt, algunas funciones avanzadas requieren más trabajo |
| **PyGUI** | Ligereza, simplicidad, uso de controles nativos | Menos funcionalidades, menor comunidad y soporte |

# Librerías

Tkinter  
Tkinter es la librería estándar de Python para interfaces gráficas, incluyéndose con la instalación básica de Python. Provee un wrapper (envoltura) para el toolkit Tcl/Tk, lo que permite al programador Python utilizar los widgets de Tk (botones, etiquetas, cuadros de texto, menús, diálogos, etc.)

Funcionalidades

* Use del módulo ttk para widgets “temáticos”, es decir, con estilos más moderno.
* Módulos auxiliares integrados: cuadros de diálogo para archivos (filedialog), selección de color (colorchooser), mensajes (messagebox), etc.
* Gestión de layout mediante distintos managers como .pack(), .grid(), .place().

Usos frecuentes

* Aplicaciones de escritorio simples o medianas donde no se necesita una interfaz muy sofisticada en gráficos o multimedia pesada.
* Herramientas internas, utilitarios, prototipos rápidos.
* Es ideal para aprendizaje, dado que está ya integrada y no requiere instalar nada extra (en muchos casos).

## Dear PyGui

Dear PyGui es una librería moderna para Python destinada a crear interfaces gráficas con alta interactividad, dinámicas, y con uso intensivo del GPU para render gráficos. Escrito en C/C++ subyacente, lo cual favorece rendimiento. Utiliza el paradigma “immediate mode” (modo inmediato), lo que lo hace eficiente para algunas clases de aplicaciones.

Características

* Permite gráficos, nodos, lienzos (“canvas”), visualizaciones, etc.
* Compatibilidad con varias plataformas: Windows, macOS, Linux, Raspberry Pi, etc.
* Licencia MIT, lo que da muchas posibilidades para uso libre/modificaciones/comercial.

Usos frecuentes

* Aplicaciones que necesitan visualizaciones interactivas, gráficos, nodos, edición visual.
* Herramientas de desarrollo, dashboards interiores, utilitarios con interfaz avanzada gráfica.
* Donde se busca rendimiento gráfico, fluidez, efectos visuales más sofisticados.

PySimpleGUI  
PySimpleGUI es una librería construida encima de otras más “pesadas” (como Tkinter, Qt, WxPython, Remi) para simplificar la creación de GUI: ofrece una interfaz más amigable, con diseño por “layout” declarativo, reduciendo mucho el código boilerplate que normalmente se requiere con Tkinter o Qt.

Funcionalidades

* “Puertos” (“ports”) para diferentes frameworks backend: el mismo código puede usar Tkinter, Qt, WxPython, o incluso interfaz web (Remi) cambiando qué “versión” de PySimpleGUI usas.
* Facilita mucho prototipado rápido. Interfaces simples, formularios, ventanas con botones, etc., se crean con pocas líneas.
* Documentación con ejemplos. Demos que muestran integración con matplotlib, OpenCV, etc.

## PyFLTK (o pyFLTK)

pyFLTK es un wrapper de Python para la librería **FLTK** (Fast Light Toolkit), que es ligera, multiplataforma, enfocada en rapidez y simplicidad. FLTK está escrita en C++ y es usada para aplicaciones que no requieren interfaces ultra complejas, pero sí que sean eficientes y ligeras.

Características

* Buena integración con OpenGL si se necesita usar gráficos, porque FLTK soporta eso.
* Ligera, tamaño reducido, menos dependencias pesadas.

Usos frecuentes

* Aplicaciones donde peso es importante (por ejemplo, en hardware limitado).
* Visualizaciones simples, utilitarios, o interfaces donde no se necesita mucho lujo gráfico sino funcionalidad.

## PyGTK / PyGObject (GTK bindings)

PyGTK era el binding para GTK+ versión 2 (GNOME). Sin embargo, está desactualizado para nuevas versiones; lo más moderno es usar **PyGObject** para trabajar con GTK+ versión 3 o 4.

GTK (GIMP Toolkit) es muy usado en entornos Linux (GNOME), con muchos widgets, soporte internacionalización, temas, etc. Las versiones actuales permiten interfaces bastante sólidas.

**Usos frecuentes**

* Aplicaciones de escritorio en Linux principalmente (aunque hay puertos/niveles de soporte para Windows/macOS).
* Aplicaciones con necesidad de apariencia nativa GTK, integración con GNOME u otras herramientas de ese ecosistema.
* Herramientas con interfaz compleja, con muchos controles, formularios, gráficos integrados, etc.

Referencias

* Python Software Foundation. (2024). Tkinter — Interfaz con el toolkit Tcl/Tk. Documentación oficial de Python. Recuperado de <https://docs.python.org/es/3.5/faq/gui.html>
* Riverbank Computing. (2024). PyQt: Python bindings for the Qt application framework. Riverbank Computing. Recuperado de <https://riverbankcomputing.com/software/pyqt/intro>
* The Qt Company. (2024). Qt for Python (PySide) — Official documentation. The Qt Company. Recuperado de https://doc.qt.io/qtforpython/
* Kivy Organization. (2024). Kivy: Cross-platform Python framework for developing multitouch applications. GitHub. Recuperado de <https://github.com/kivy/kivy>
* Dear PyGui Developers. (2024). Dear PyGui Documentation. Read the Docs. Recuperado de <https://dearpygui.readthedocs.io/>
* GeeksforGeeks. (2024). Introduction to Kivy: A cross-platform Python framework. GeeksforGeeks. Recuperado de <https://www.geeksforgeeks.org/introduction-to-kivy/>
* ActiveState. (2023). Python GUI programming: wxPython vs. Tkinter. ActiveState Blog. Recuperado de <https://www.activestate.com/blog/python-gui-programming-wxpython-vs-tkinter/>
* Stack Overflow. (2023). What is the difference between Tkinter, WxWidgets, PyQt, and PySide? Stack Overflow. Recuperado de <https://stackoverflow.com/questions/75845338/>
* Lutz, M. (2011). Programming Python (4th ed.) — Section 8.2: Python GUI Development Options. O’Reilly Media. Recuperado de <https://flylib.com/books/en/2.726.1.72/1/>
* PythonHint. (2024). PyQt vs. PySide comparison: Licenses, performance, and documentation. PythonHint. Recuperado de <https://pythonhint.com/post/7583871706064394/pyqt-vs-pyside-comparison>