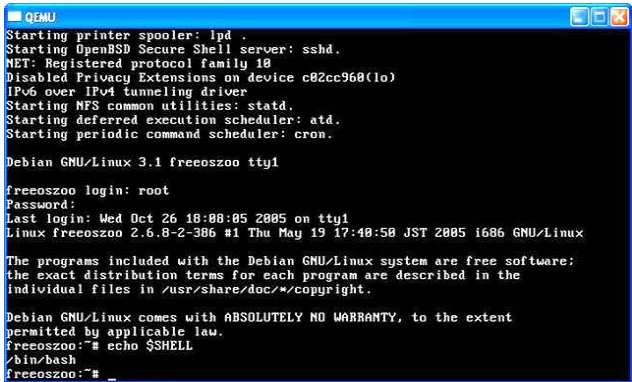

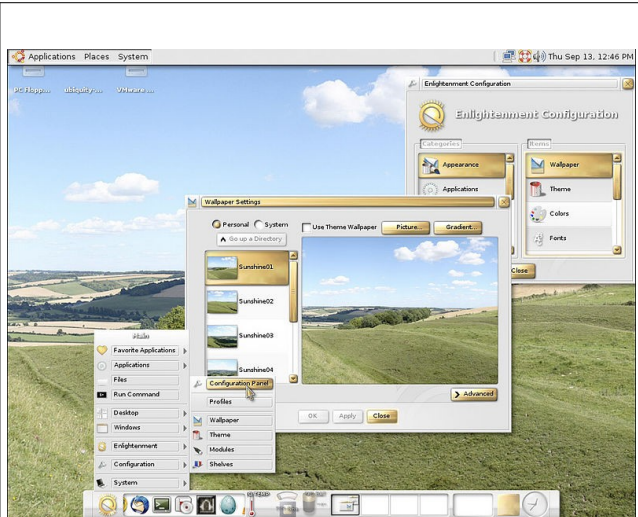


➤ Introducción

La **interfaz gráfica de usuario** es el sistema interactivo que posibilita, a través del uso y la representación del lenguaje visual, una interacción amigable con un sistema informático. Están formado un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. El desarrollo de interfaces es bastante costoso, al principio suponía más del 50% del desarrollo de la aplicación.

<p>A finales de la década de los sesenta aparecen las primeras interfaces gráficas de usuario tal como las entendemos hoy en día. Previamente, las interfaces de usuario eran simples CLI (Interfaces Command Line) donde se introducían las órdenes mediante comandos con sus respectivas opciones. Su desventaja es que obliga a memorizar las órdenes y sus atributos.</p>	
<p>Posteriormente, aparecieron los primeros IU o Interfaces de Usuario con menús jerárquicos como por ejemplo Windows 3.1. Aunque facilita mucho el uso del sistema, la estructura jerárquica obliga a realizar varios pasos de navegación hasta la función deseada.</p>	

En los 80' aparecieron las interfaces **WIMP** (*Windows, Icons, Menus and Pointing device*), es decir, **ventanas, iconos, menus y punteros**. Se mejoraron con versiones con simulación 3D y la manipulación se hace directa mediante la representación visual de los objetos y la aparición de los eventos. Su origen está en una interfaz desarrollada por **Xerox**.



➤ Librerías Gráficas

Una biblioteca es un **conjunto subprogramas** con una interfaz bien definida que al ser invocados son utilizados por otro software. Las bibliotecas contienen código y datos, que proporcionan **servicios a programas independientes sin relación entre ellos y que pasan a formar parte de estos**. Esto permite que el código y los datos se compartan y puedan modificarse de forma modular. Ejecutables y bibliotecas hacen referencias (llamadas enlaces) entre sí a través de un proceso conocido como enlace realizado por un software llamado **enlazador**.

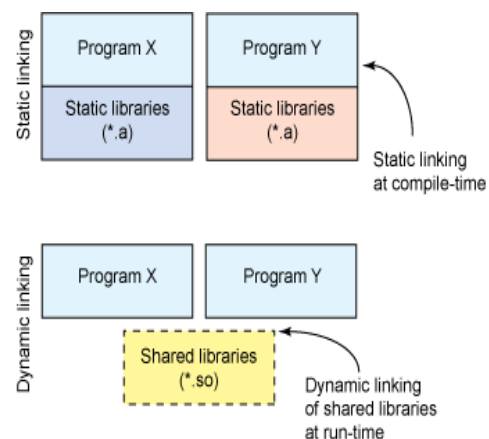
La principal ventaja de las bibliotecas es que descarga al programador de tener que diseñar módulos que ya son comunes a otros programas. De hecho, muchas de las librerías ya están incorporadas al sistema operativo y ser utilizadas por el software

Apunte histórico

- 1981 – Xerox Star (ventanas y 1ª aparición del ratón)
- 1984 – Apple Macintosh, X Windows (popularización del ratón)
"There is no evidence that people want to use these things"
- 1984/7 – X Window System (X11, MIT, separado del SO y WM)
- 1985 – MS Windows (integrado en el SO e incluye WM)
- Finales de los 90 – Interfaces Web...

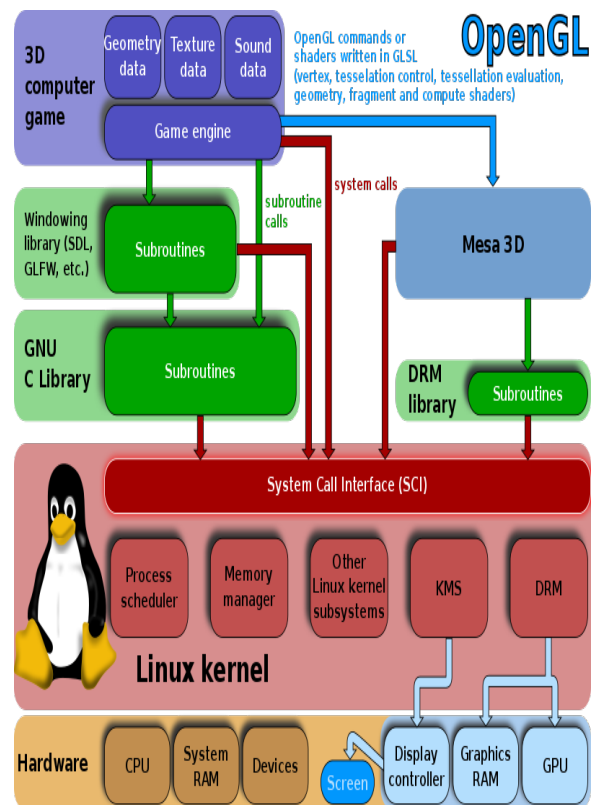
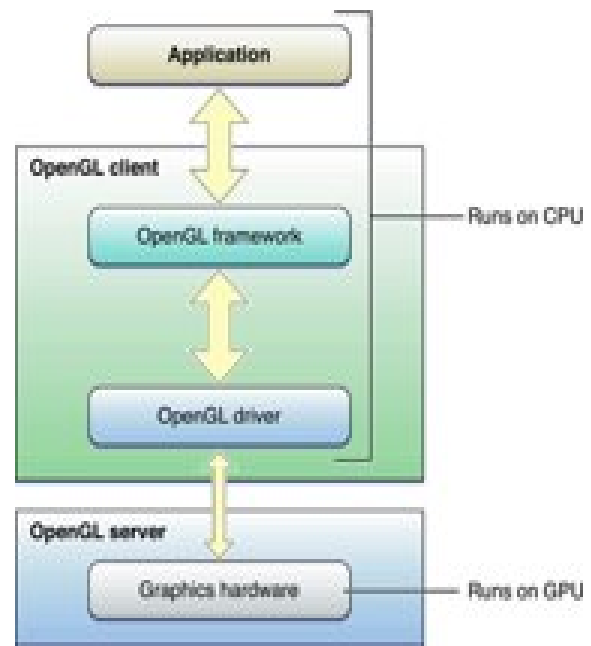
que instalamos posteriormente. Un ejemplo, sería el módulo de impresión que es utilizado por diferentes programas que lo invoquen.

- **Librería estática:** es aquella que se **enlaza en tiempo de compilación**. La ventaja de este tipo de enlace es que hace que un programa **no dependa de ninguna biblioteca** (puesto que las enlazó al compilar) y **facilita sus distribución**. Su inconveniente es que los **programas son más pesados y menos flexibles** a la hora de modificar su código. El enlazado permite al programador y al propio sistema operativo dividir un programa en varios archivos llamados módulos.
- **Librería dinámica:** es aquella enlazada cuando un determinado programa se ejecuta. La ventaja de este tipo de enlace es que el **programa es más liviano**, y que **evita la duplicación de código**. Las bibliotecas de enlace dinámico, o **bibliotecas compartidas**, suelen encontrarse en directorios específicos del sistema operativo, por ejemplo, en Linux están en */lib*. En Windows las librerías dinámicas se denominan DLL (Dynamic-Link Library).



Los sistemas operativos (SO) modernos proporcionan bibliotecas que implementan los servicios más comunes del sistema. Por lo tanto, se han convertido en una "materia prima" que cualquier aplicación espera obtener del SO con el objetivo de que se evite el tener que reescribir el mismo módulo en diferentes aplicaciones dando como resultado a una mayor producción en el desarrollo de software.

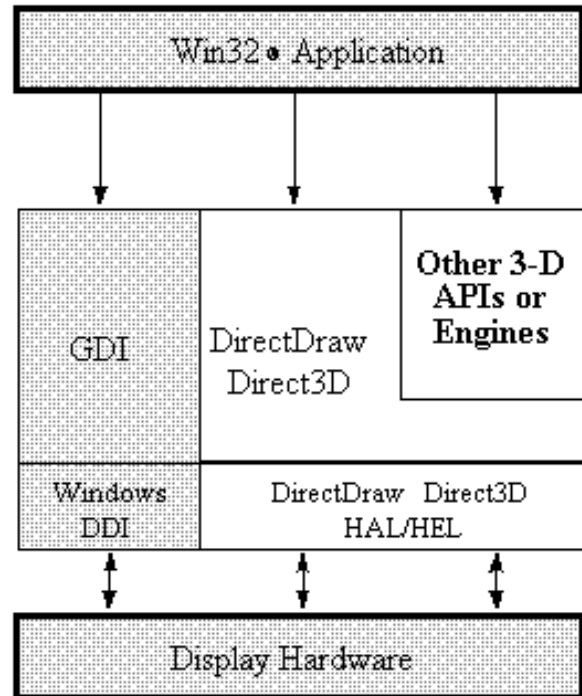
OpenGL (Open Graphics Library) es una especificación estándar que define una API **multilenguaje** y **multiplataforma** para escribir aplicaciones que produzcan gráficos 2D y 3D. La interfaz consiste en más de 250 funciones diferentes que pueden usarse para dibujar escenas tridimensionales complejas a partir de primitivas geométricas simples, tales como **puntos, líneas y triángulos**. Fue desarrollada originalmente por Silicon Graphics Inc. (SGI) y se usa ampliamente en CAD, realidad virtual, representación científica, visualización de información simulación de vuelos y desarrollo de videojuegos, donde compite con Direct3D en plataformas Microsoft Window. El funcionamiento básico de OpenGL consiste en **aceptar primitivas tales como puntos, líneas y polígonos, y convertirlas en píxeles**. Este proceso es realizado por una pipeline gráfica conocida como *Máquina de estados de OpenGL*. El sucesor de OpenGL se llama Vulkan.



DirectX-Direct3D: Es parte de DirectX (conjunto de bibliotecas para multimedia), propiedad de **Microsoft**. Consiste en una API para la programación de gráficos 3D disponible tanto en sistemas Windows de 32 y 64 bits, y sus consolas Xbox. Direct3D está compuesto por dos grandes APIs: el modo retenido y el modo inmediato.

El modo inmediato da soporte a todas las primitivas de procesamiento 3D que permiten las tarjetas gráficas (luces, materiales, transformaciones, control de profundidad, etc).

El modo retenido, construido sobre el anterior, presenta una abstracción de nivel superior ofreciendo funcionalidades preconstruidas de gráficos como jerarquías o animaciones lo que facilita los desarrollos.



GTK "GIMP Tool Kit" es una biblioteca que contiene los objetos y funciones para crear la interfaz gráfica de usuario. Maneja widgets como ventanas, botones, menús, etiquetas, deslizadores, pestañas... y **es usado por Gnome**. GTK se ha diseñado para permitir programar con lenguajes como C, C++, C#, Java, Ruby, PHP, Perl, o Python. Está formada por:

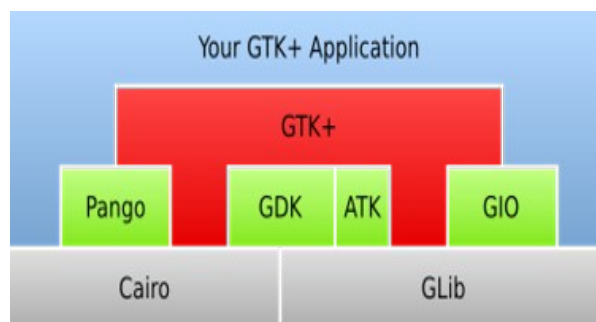
- **GLib.** Biblioteca de bajo nivel estructura básica de GTK+ y GNOME. Proporciona manejo de estructura de datos para C, portabilidad, interfaces para funcionalidades de tiempo de ejecución como ciclos, hilos, carga dinámica o un tema de objetos.

- **GTK.** Biblioteca la cual realmente contiene los objetos y funciones para crear la interfaz de usuario. Maneja widgets como ventanas, botones, menús, etiquetas, deslizadores, pestañas, etc.
- **GDK.** Biblioteca que actúa como intermediario entre gráficos de bajo nivel y gráficos de alto nivel.

ATK. Biblioteca para crear interfaces con características de una gran accesibilidad muy importante para personas discapacitadas o minusválidos. Pueden usarse utilerías como lupas de aumento, lectores de pantalla, o entradas de datos alternativas al clásico teclado o ratón.

Pango. Biblioteca para el diseño y renderizado de texto, hace hincapié especialmente en la internacionalización. Es el núcleo para manejar las fuentes y el texto de GTK+2.

Cairo: Permite dibujar las animaciones y sombras de gráficos.



QTK: es una biblioteca multiplataforma usada para desarrollar aplicaciones con interfaz gráfica de usuario, así como también para el desarrollo de programas sin interfaz gráfica, como herramientas para la línea de comandos y consolas para servidores. **Usada preferentemente por escritorio como KDE.** Qt utiliza el lenguaje de programación **C++** de forma nativa, adicionalmente puede ser utilizado en varios otros lenguajes de programación a través de bindings. También es usada en sistemas informáticos empotrados para automoción, aeronavegación y aparatos domésticos como frigoríficos.

wxWINDOW: Es una biblioteca de clases para **C++ y Python**, que permite el desarrollo de aplicaciones con interfaces gráficas de usuario de una manera rápida y sencilla. Su principal característica es que es **multiplataforma**. Se distribuye bajo licencia *wXWindows Library License*, que es similar a la GNU Library General Public License pero que además permite usar la biblioteca para desarrollos comerciales (ya sean aplicaciones o modificaciones de la propia biblioteca), siempre y cuando estos desarrollos comerciales no usen ningún código distribuido bajo alguna licencia GNU

Estos son los tipos principales de librerías gráficas existentes en el mundo Linux y Windows.

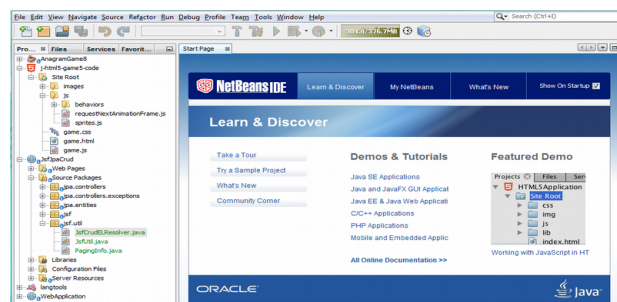
➤ Entornos de Desarrollo Integrado

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un **editor de código** a ser posible **autocompletador**, **gestor de proyecto**, deseable **soporte para múltiples lenguajes**, recomendable con un autocompletador, un **compilador**, un **depurador** y un **constructor de interfaz gráfica (GUI)**. Si tiene todos estos elementos la productividad aumenta exponencialmente. Ejemplos de los más influyentes son:

Eclipse es un entorno de desarrollo integrado de código abierto y multiplataforma con soporte para muchos lenguajes, especialmente para Java. Tiene un plugin para python PyDev.



NetBeans es un IDE multiplataforma con licencia GPL2. Se usa para desarrollar desde aplicaciones de escritorio, aplicaciones web y móvil, Tiene bastantes módulos que al instalarlos se extiende su funcionalidad y herramientas para otros lenguajes especialmente Java. Tiene plugins para Python.



Visual Studio de pago creado por la Microsoft aunque existen versiones gratuitas para estudiantes, se puede desarrollar desde aplicaciones de escritorio, web, móvil... pero solo se ejecuta en el SO Windows, bajo la plataforma .NET. Permite programar en Basic, Java, C# y C++ entre otros. Permite instalar extensiones para Python.



Luego tenemos los **IDES** específicos para Python en Linux. No significa que solo se programe en Python con ellos, son multilenguajes pero son utilizados y recomendados por parte de la comunidad de Python.

- **BeeWare** es una caja de herramientas de ayuda para desarrollar y depurar software en Python. La gran diferencia de con un IDE es que cada herramienta de esa caja puede usarse de forma independiente del resto.
- **Boa Constructor** es un IDE con una interfaz gráfica de usuario incorporada (GUI) para wxPython. Incluye un inspector de objetos, jerarquías de herencia, depurados y ayuda integrada.
- **Pycharm:** IDE creado por JetBrains. Dentro. De los más influyentes. Cuenta cientos de funciones. Algunas de sus características son: Integración con framework como Django o Web2Py, autocompletado, resaltador de sintaxis, herramienta de análisis, refactorización, depurador avanzado de Python, compatibilidad con SQLAlchemy (ORM), Google App Engine o Cython, sistema de control de versiones como Git, CVS y Mercurial.

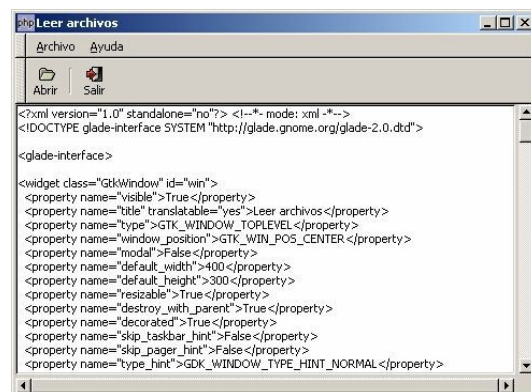
Hay bastantes más, algunos son proyectos semiabandonados, otros siguen en activo como son Anjuta, IDE para C++, Java y Python, pero no es cuestión de dar una lista muy larga. En nuestro caso vamos a trabajar con Glade + Python en nuestras actividades usando como herramienta NetBeans, aunque cuyos plugins para Python no sean los mejores si los comparamos con los de Eclipse y Pycharm.

- **Glade:** es un diseñador de interfaces gráficas bajo licencia GPL que utiliza **GTK/Gnome**. Es independiente del lenguaje de programación. Al diseñar una interfaz gráfica se genera un **fichero XML**. El fichero XML generado tiene un formato llamado **GtkBuilder** para almacenar los elementos de las interfaces diseñadas. Estos archivos cuando el SO lo lee en tiempo de ejecución mediante el objeto GtkBuilder de GTK+ genera la UI. GladeXML era el formato que se usaba en conjunto con la biblioteca libglade (ambos obsoletos en favor de GtkBuilder).
- **Python:** es un lenguaje multiplataforma interpretado. Su sintaxis, sencilla, hace que se genera un código muy legible que es uno de sus principios fundamentales. Soporta programación orientada a objetos, programación imperativa y programación funcional pero en menor medida, con herramientas como lambda, reducir, filter y map entre otras. Otras características:
 - Uno de las frameworks más utilizados para desarrollo para web, Django, está escrito en python.
 - Tiene su propio repositorio para terceros **pypi**.
 - El índice de *Programación Tiobe*, sitúa a python, tras Java y C/C++ como el lenguaje más utilizado en 2018 desbancando entre otros a javascript, visualnet y php.
 - En contra, no es buen lenguaje para desarrollo de juegos en 3D y móviles, de momento, aunque tiene la herramienta pygame que está evolucionando.

```
>>> class Myobj1:
    def __init__(self, i, j):
        self.i = i
        self.j = j
    def muestre(self):
        print 'Myobj1: i= ',self.i, ' j= ',self.j
    def saludo(self):
        print 'Hola, soy un objeto'
        print 'Puedas usar mis Atributos y Metodos'

>>>
```

Código Python



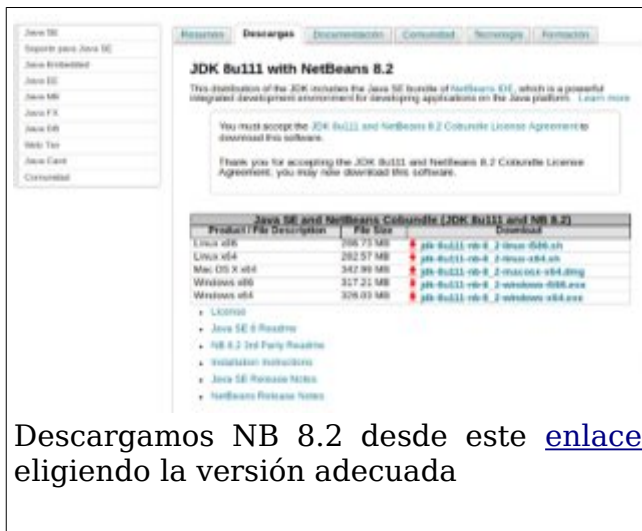
Fichero XML de una GUI en Glade

A continuación vamos a realizar una serie de actividades que consistirán en ver las diferentes opciones existentes y que utilizaremos para programar interfaces gráficas con python.

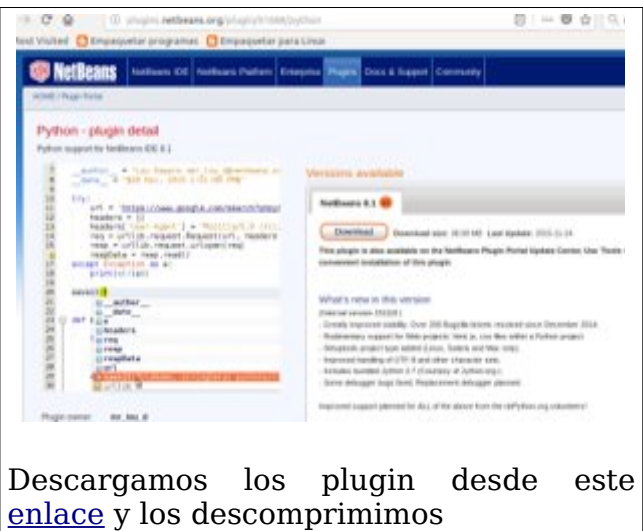
ACTIVIDAD RESUELTA

Opción1. Vamos a llevar a cabo la configuración del IDE Netbeans (NB), concretamente la versión 8.2 con un diseñador de ventanas como Glade. En la web oficial de NB la documentación existente para los plugins de python hace referencia a la versión 8.1 de NB pero que suele funcionar sin problemas para la versión 8.2. En caso contrario, se usa la versión 8.1 que es prácticamente idéntica.

- Herramientas:
 - Ubuntu. Versión 16.04 o si es posible la versión 18.04
 - NB versión 8.2 con los plugins de Python. En su defecto, la versión 8.1
 - Glade. Actualmente la versión estable es la 3.20



Descargamos NB 8.2 desde este [enlace](#) eligiendo la versión adecuada



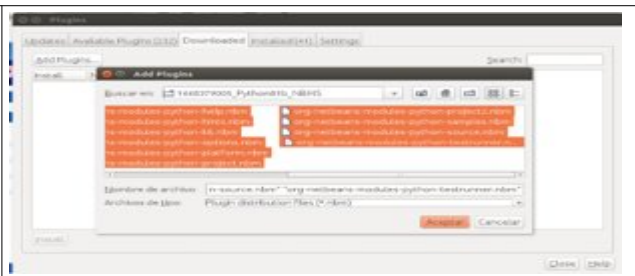
Descargamos los plugin desde este [enlace](#) y los descomprimos

A veces ocurre el siguiente error: *“failed to load module canberra-gtk-module”*. Esto tiene fácil solución, simplemente se instala lo siguiente:

```
$ sudo apt install libcanberra-gtk-module
```

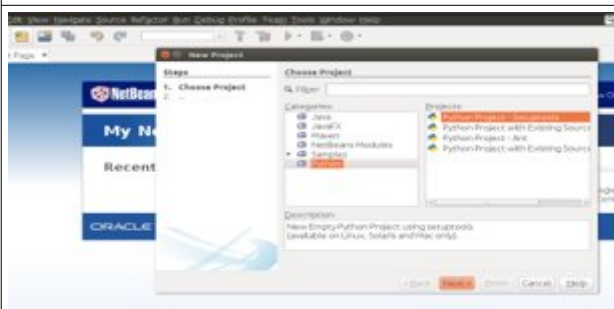


Paso siguiente: ejecutar el script con sudo. Según el equipo puede llevar más o menos tiempo.

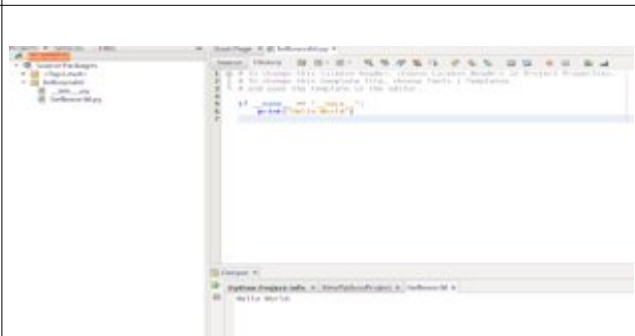


Ejecutamos NB: *Herramientas* → *Plugins* → *Add Plugins*

Seleccionamos todos e **Install** (no hacer caso de los warning)



Al reiniciar NB. Vemos que nos da la opción de crear un proyecto en Python2.7. También se puede realizar con Python 3. Esta opción es mejor porque nos evita los problemas de la ñ o los acentos



Ejecutamos y observamos que todo va bien

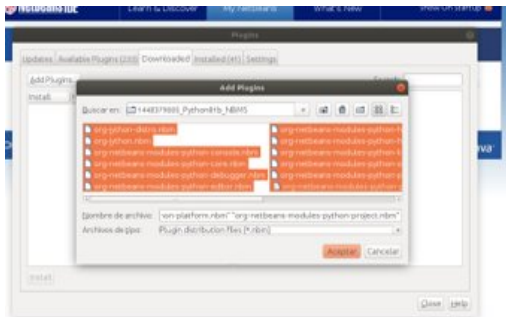
Otro error común es que cuando intentamos lanzar NB 8.2 nos muestra este aviso y no se carga.



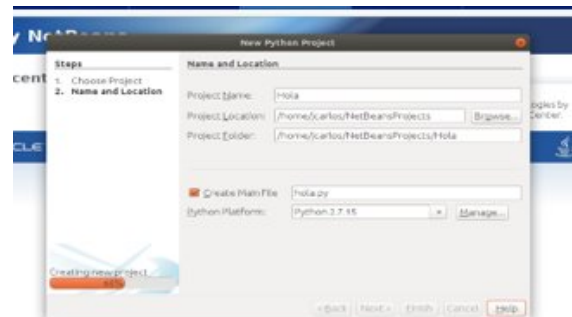
Para resolverlos hacemos lo siguiente, abrimos el fichero *netbeans.conf* y añadimos la línea que se indica:

```
jcarlos@portatil: ~
Archivo Editar Ver Buscar Terminal Ayuda
jcarlos@portatil:~$ sudo find / -name 'netbeans.conf'
find: '/run/user/1000/gvfs': Permiso denegado
/home/jcarlos/netbeans-8.2/etc/netbeans.conf
jcarlos@portatil:~$ sudo nano /home/jcarlos/netbeans-8.2/etc/netbeans.conf
```

```
GNU nano 2.9.3 /home/jcarlos/netbeans-8.2/etc/net
# It can be overridden on command line by using --jdkhome
# Be careful when changing jdkhome.
# There are two NetBeans launchers for Windows (32-bit
# installer points to one of those in the NetBeans app
# based on the Java version selected at installation t
#
netbeans_jdkhome="/usr/lib/jvm/java-8-oracle"
# Additional module clusters:
```

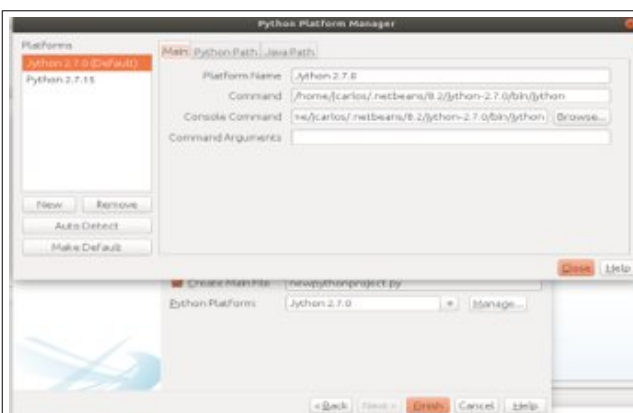


Ahora ya podemos descargar los plugins desde aquí, descomprimir para con *Herramientas* → *Plugin*. Instalamos y volvemos a abrir

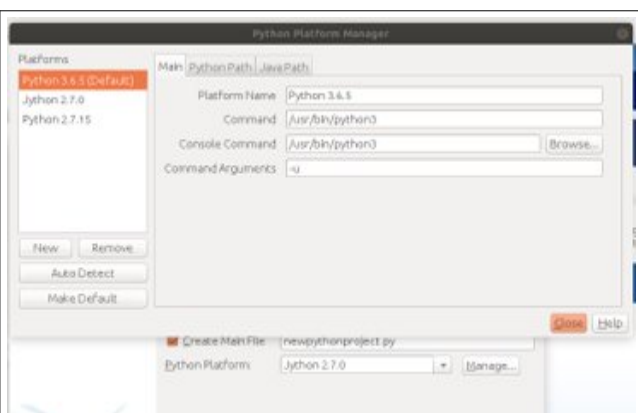


Creamos un nuevo proyecto y probamos.

Si queremos usar Python3 que nos resuelve, entre otros y como hemos dicho, el problema de *tildes* y *ñ*, hacemos lo siguiente.



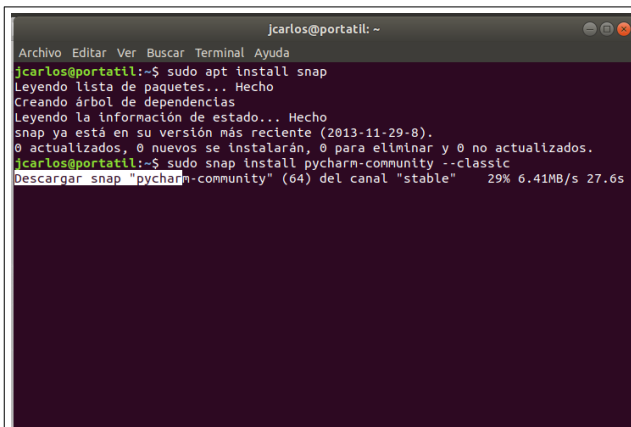
Al crear un nuevo proyecto pulsamos MANAGE y NEW.



Buscamos */usr/bin/python3* y lo establecemos por defecto.

Opcion 2. Una herramienta con gran influencia en el mundo Python es **Pycharm**. Existe una versión gratuita **Community**. La versión que instalaremos dependerá de la versión de Ubuntu. También existe una versión para Windows.

Empezaremos por Ubuntu 18.04 aunque también es válido para la 16.04



```
jcarlos@portatil: ~  
Archivo Editar Ver Buscar Terminal Ayuda  
jcarlos@portatil:~$ sudo apt install snap  
Leyendo lista de paquetes... Hecho  
Creando árbol de dependencias  
Leyendo la información de estado... Hecho  
snap ya está en su versión más reciente (2018-11-29-8).  
0 actualizados, 0 nuevos se instalarán, 0 para eliminar y 0 no actualizados.  
jcarlos@portatil:~$ sudo snap install pycharm-community --classic  
Descargar snap "pycharm-community" (64) del canal "stable" 29% 6.41MB/s 27.6s
```

Para ello ejecutamos los siguientes comandos:

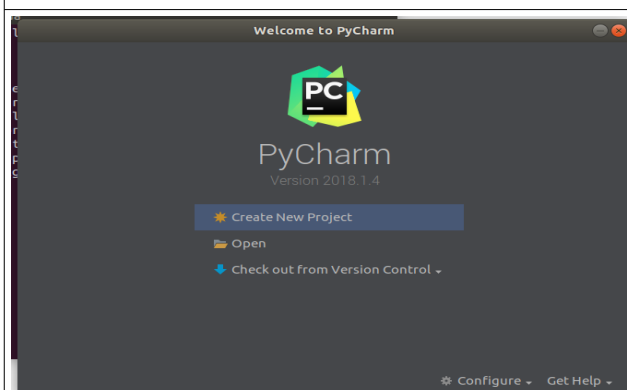
```
$sudo get install snap snap-zdg-open
```

```
$sudo snap install pycharm-community --classic
```

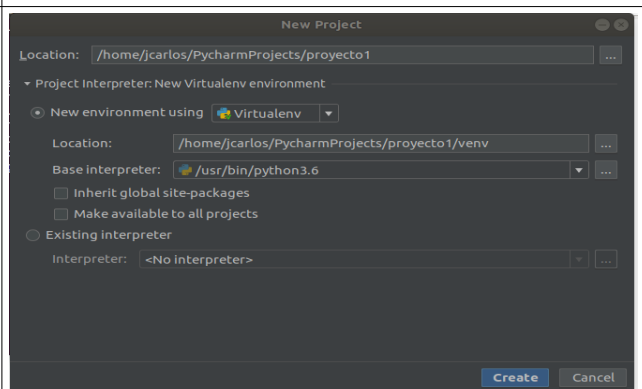


Para lanzarlo bien en modo gráfico o bien ejecutando

```
$pycharm-community
```

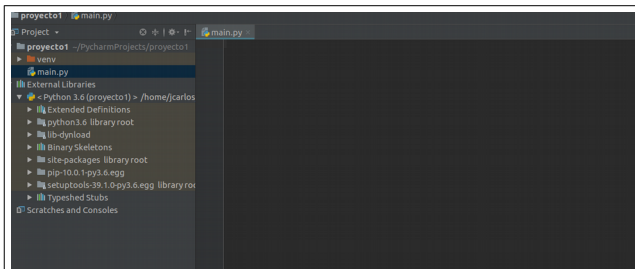


Ahora nos va a solicitar algunos datos y configuraciones a gusto del desarrollador. También en *File* → *Settings* podemos “tunear” el IDE posteriormente.



Creamos un nuevo proyecto y le damos nombre, por ejemplo, *proyecto1*

A partir va cargando la configuración y según la máquina puede tardar algunos segundos.



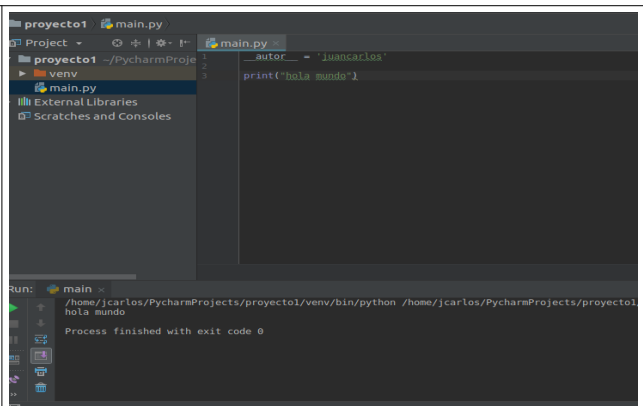
Nos situamos en el nombre del proyecto y botón derecho

Proyecto1 → *New File*

Y escribimos el código siguiente:

```
__autor__ = 'juancarlos'

print ("Hola Mundo")
```



Y ejecutamos

A partir de ahora podemos elegir el IDE que deseemos. Sin embargo, a pesar de la gran aceptación de Pycharm por parte de la comunidad de Python por su herramienta de *autocompletar* para el lenguaje he observado bastante problemas a la hora de trabajar con la librería Gtk por ello aconsejo usar NB aunque más limitado en Python no presenta tantos problemas a la hora de manejar las librerías gráficas.

Python, como se comentó en apartado anteriores puede ser codificado en otros IDE's, incluso en Visual Net que contiene los plugins necesarios. Sin embargo, como utilizaremos la herramienta Glade, cuyo desarrollo principal es en Linux, será este SO la base sobre la que desarrollaremos la parte práctica de la materia.

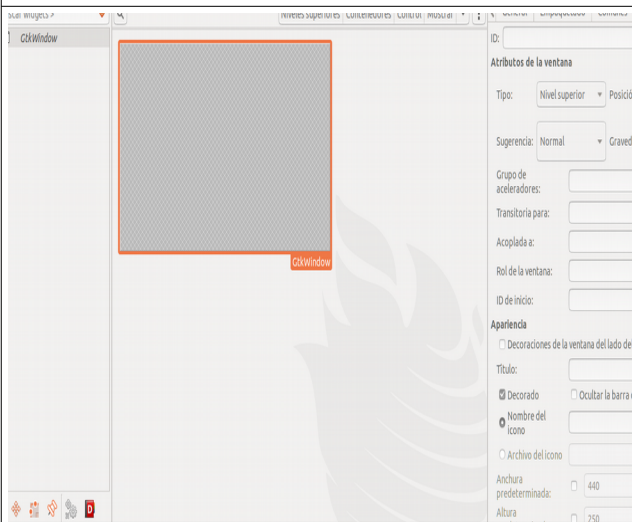
ACTIVIDAD RESUELTA

Ahora que ya tenemos un IDE listo para ser usado pasamos crear nuestro primer interfaz gráfico pero para ello debemos descargar el IDE para diseño de GUI Glade para Ubuntu.

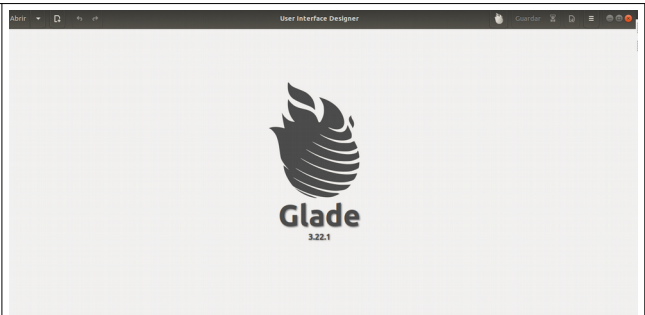
```
jcarlos@portatil:~$ sudo apt install glade
[sudo] contraseña para jcarlos:
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes adicionales:
  devhelp devhelp-common libatk1.0-doc libdevhelp-3-5 libgladeui-2-6
  libgladeui-common libglib2.0-doc libgtk-3-doc libpango1.0-doc
Paquetes sugeridos:
  libgtk2.0-doc
Se instalarán los siguientes paquetes NUEVOS:
  devhelp devhelp-common glade libatk1.0-doc libdevhelp-3-5 libgladeui-2-6
  libgladeui-common libglib2.0-doc libgtk-3-doc libpango1.0-doc
0 actualizados, 10 nuevos se instalarán, 0 para eliminar y 0 no actualizados.
Se necesita descargar 7.418 kB de archivos.
Se utilizarán 61,7 MB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n]
```

Ejecutamos:

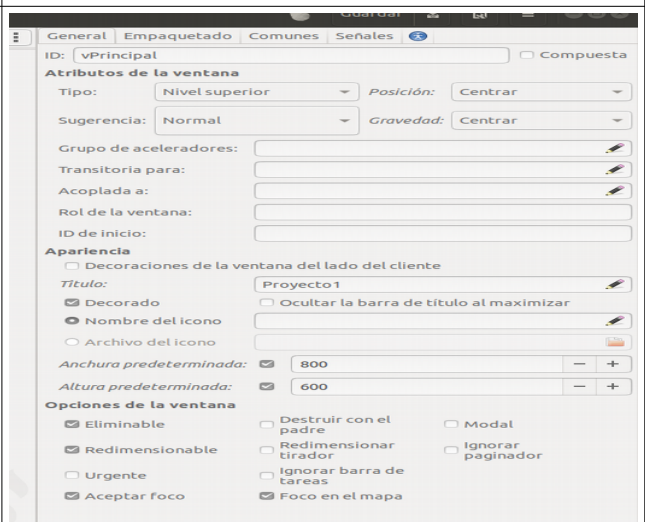
```
$sudo apt install glade
$sudo apt install python-pip python3-pip
$sudo apt install python-gi python3-gi
```



Creamos un nuevo proyecto y en *niveles superiores* añadimos una ventana, es decir, *GtkWindow*



Vamos a construir una sencilla interfaz con un botón y un label de forma que al pulsar el botón en el texto de la etiqueta aparecer el mensaje *Hola Mundo*. Lanzamos Glade

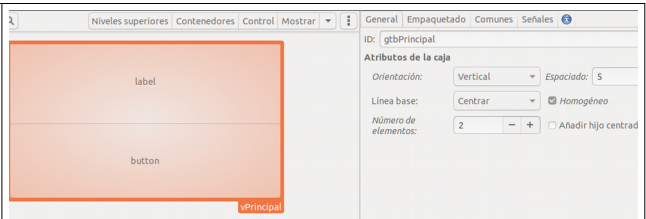


Cambiamos algunos datos en *General* y en *Comunes*:
 Título: Proyecto1
 ID: vPrincipal
 Posición y Gravedad: Centrar
 Tamaño: 600 x 800

En señales elegimos *GtkWidget* → *Destroy*. Observamos que automáticamente nos aparece el nombre, *on_vPrincipal_destroy*, que será el que identifique al evento que codificaremos posteriormente.



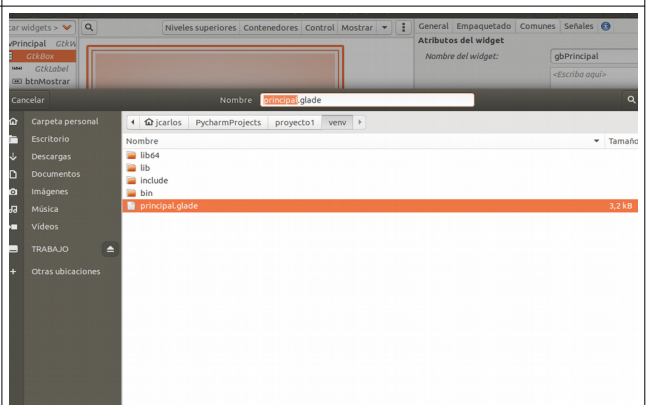
En el botón *Contenedores* elegimos *GtkBox* configuramos para dos filas o elementos en *General*



En la superior con el botón *Mostrar* elegimos un *Label* o etiqueta. En el inferior desde *Control* elegimos un *Botón*.



Ahora podemos poner márgenes, quitar el texto de label y cambiar el título de los botones, alineación centrar en el botón, entre otras cosas. En *señales* del botón escribimos otro evento *GtkButton* → *on_btnMostrar_clicked*



Vamos a guardar el resultado en la misma carpeta del Proyecto1 que creamos en la actividad anterior con el nombre *principal.glade*

La mejor manera de conocer los diferentes atributos y propiedades de los widgets es ir experimentando con ellos. Las GUI de Linux no están tan desarrolladas como las de Windows donde las herramientas permiten disposiciones en la ventana gráfica mucho más elaboradas. Aún así, se obtiene resultados bastante aceptables.

Ahora que tenemos la interfaz gráfica vamos a escribir el código necesario. Fijemonos que tenemos dos eventos. El primero *on_vPrincipal_destroy* nos permitirá cerrar la ventana y el proceso asociado pulsando la X de la mismas. El segundo que

es el *on_btnMostrar_clicked* hará que en la etiqueta o *label* nos muestre el mensaje "HOLA MUNDO".

Lancemos pues NB y creamos o abrimos el proyecto1 y en *main.py* codificamos el código siguiente.

```
import gi

gi.require_version('Gtk', '3.0')

from gi.repository import Gtk

class Hola:

    def __init__(self):

# iniciamos la libreria GTK

        b= Gtk.Builder()

        b.add_from_file("principal.glade")

# cargamos aquellos widgets con algún evento asociado o son referenciados

        self.vprincipal = b.get_object("vPrincipal")

        self.btnhola = b.get_object("btnMostrar")

        self.lblhola = b.get_object("lblHola")

        # el diccionario de eventos

        dic = {'on_btnMostrar_clicked': self.mostrar, 'on_vPrincipal_destroy':
self.salir, }

        # conectamos todo y mostramos

        b.connect_signals(dic)

        self.vprincipal.show()
```

ahora vienen las funciones

```
def mostrar(self, widget, data=None):
```

```
    self.lblhola.set_text("HOLA MUNDO")
```

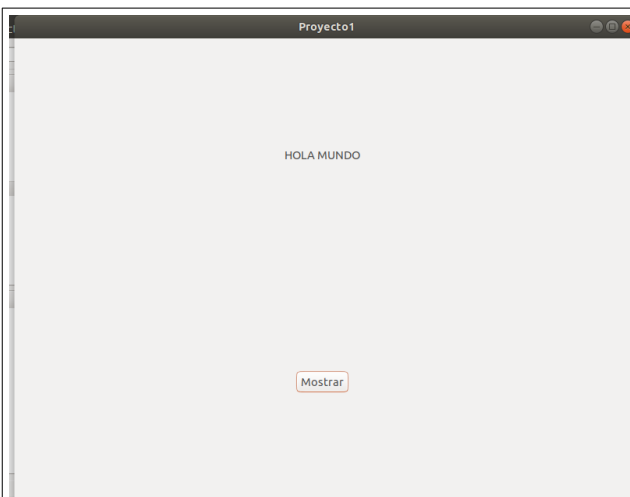
```
def salir(self, widget, data=None):
```

```
    Gtk.main_quit()
```

```
if __name__ == "__main__":
```

```
    main = Hola()
```

```
    Gtk.main()
```



El resultado al pulsar el botón

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Generated with glade 3.22.1 -->
<interface>
  <requires lib="gtk+" version="3.20"/>
  <object class="GtkWindow" id="vPrincipal">
    <property name="can_focus">False</property>
    <property name="margin_left">5</property>
    <property name="margin_right">5</property>
    <property name="margin_top">5</property>
    <property name="margin_bottom">5</property>
    <property name="title" translatable="yes">Proyecto1</property>
    <property name="window_position">center</property>
    <property name="default_width">800</property>
    <property name="default_height">600</property>
    <property name="gravity">center</property>
    <signal name="destroy" handler="on_vPrincipal_destroy" swapped="no"/>
    <child>
      <placeholder/>
    </child>
    <child>
      <object class="GtkBox">
        <property name="name">gbPrincipal</property>
        <property name="visible">True</property>
      </object>
    </child>
  </object>
</interface>
```

Y echemos un vistazo al fichero *principal.glade* con el editor. Nos resulta un fichero con formato XML

ACTIVIDADES PROPUESTAS

Diseña una aplicación con una ventana y un botón *Abrir* que una vez lanzada muestra otra ventana con un mensaje de bienvenida y un botón con *Salir* que cierre la ventana secundaria.

Tamaño de la ventana principal 600x400

Tamaño de la ventana secundaria 300x200

Ambas estarán centradas

La ventana secundaria sera **transitoria** de la ventana principal. Esto quiere decir que si cerramos la ventana principal, la ventana secundaria también lo hace.