

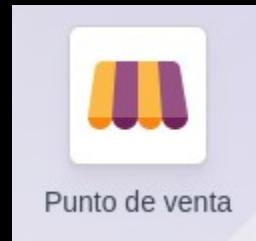
SISTEMAS DE GESTIÓN EMPRESARIAL

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

**Punto de venta
y
Fabricación
en Odoo**

Configurando el punto de venta de la empresa

En este tema, continuaremos con el ejemplo de la empresa de muebles anterior que, tal y como comentamos, dispone de tiendas físicas donde los clientes hacen sus compras. Para gestionar las tiendas, haremos uso del módulo de **Punto de venta (PdV)**.



Este módulo requiere tener instalados los de **Contabilidad** e **Inventario**, pero estos ya los tenemos instalados del tema anterior. Además, para realizar algunos casos prácticos con los que nos encontraremos, debemos activar el **checkbox de Contabilidad** en la configuración del módulo de PdV *:

Marcando el checkbox de contabilidad

A screenshot of a software interface showing configuration options for the POS. The title 'Facturas y recibos' is at the top. There are four checkboxes listed: 'Encabezado y pie de página' (unchecked), 'Reimprimir Recibo' (unchecked), 'Impresión automática del recibo' (unchecked), and 'Contabilidad' (checked). A red box highlights the 'Contabilidad' checkbox. At the bottom right, there are links for 'Diario de factura' and 'Facturas de cliente (EUR)'.

Este checkbox nos permite imprimir facturas a solicitud del cliente

* En la última versión online de prueba (dic/2023) este checkbox no existe, pero ya está incluido Contabilidad.

Desde esa misma pantalla de configuración, se pueden configurar muchas más opciones del PdV, como la interfaz, los dispositivos conectados (TPVs, impresoras de tickets...), el método de pago (efectivo, banco...), etc.

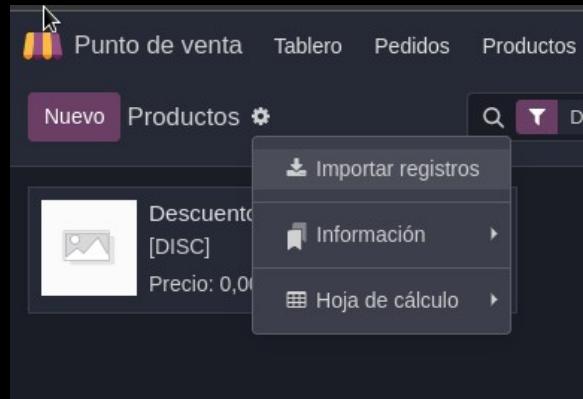
Importar productos al punto de venta

Supongamos que desde el almacén central, nos envían un **listado con los productos** que se quieren vender en la tienda donde nos encontramos. Concretamente, el siguiente **fichero Excel**:

Este está formado por las siguientes columnas:

Columnas del fichero Excel

Para **importar los productos** en nuestra tienda, basta con entrar en el **módulo de PdV** pasar al Producto y pulsar sobre el botón “**Importar**”.



A continuación, tras pulsar sobre el botón de “**Cargar fichero**”, buscamos nuestro fichero y lo abrimos.

Es entonces cuando se nos **muestra la tabla con todos los datos** del fichero. Antes de pulsar el botón de “Importar”, podemos hacer una prueba pulsando el botón “**Test**” para comprobar que no hay ningún problema.

Probando que los datos están correctos antes de importarlos.

The screenshot shows the Odoo Product Import wizard. On the left, under 'Archivo importado', there is a file icon and the path 'listado_de_productos.csv'. A checked checkbox 'Utilizar la primera fila como encabezado' (Use the first row as header) is selected. On the right, a large blue banner says 'Todo parece correcto.' (Everything seems correct). Below it, a table maps columns from the CSV to Odoo fields:

Columna del archivo	Campo de Odoo	Comentarios
Extrenal ID product_template_1	Para imp... ▾	
Name Acoustic Block Screens	Ab N ✖ ▾	
Product Type Product	▼ T. ✖ ▾	

Una vez que nos cercioramos que todo está bien, pulsamos sobre importar:

Visualizando algunos de los productos importados

The screenshot shows the Odoo Products list view. At the top, there is a search bar with filters 'Disponible en TPV' and 'Buscar...', and a page indicator '1-7 / 7'. The list displays six products:

Acoustic Block Screens [FURN_6666] Precio: 350,00 € A mano: 0,00 Unidades	Descuento [DISC] Precio: 0,00 € A mano: 0,00 Unidades	Desk Organizer [FURN_0001] Precio: 20,00 € A mano: 0,00 Unidades
Desk Pad [FURN_0002] Precio: 35,00 € A mano: 0,00 Unidades	Drawer [FURN_8855] Precio: 90,00 € A mano: 0,00 Unidades	Flipover [FURN_9001] Precio: 80,00 € A mano: 0,00 Unidades
LED Lamp [FURN_0003] Precio: 200,00 € A mano: 0,00 Unidades		

Una vez creados, podemos editarlos. Sobre todo, algo útil es **informar de la categoría** para luego buscarlo más fácilmente:

Editando el producto cargado

Punto de venta Tablero Pedidos Productos Informes Configuración

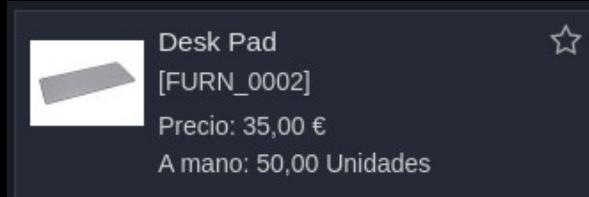
Nuevo Productos [FURN_0001] Des... Extra Prices Documentos A Mano 0,00 Unidades Más 3 / 7 < >

Tipo de producto ?	Almacenable	Precio de venta ?	€20,00	(= 24,20 € impuestos incluidos)
Política de Facturación ?	Cantidades pedidas	Impuestos del cliente ?	21% G (Bienes) x	
Los productos almacenables son artículos físicos para los que se gestiona el nivel de inventario.				
Puede facturarlas antes de que se entreguen.				
Coste ?	€83,00	Categoría de producto	Varios	
Referencia interna	FURN_0001	Código de barras		
Etiquetas de la plantilla del producto				
→ Configurar etiquetas				

Venta de productos en el punto de venta

Para esta simulación, hemos actualizado la cantidad del producto "Desk Pad" a 50 unidades:

Cantidad actualizada



En el siguiente vídeo, comprobaremos lo intuitivo y fácil que es realizar una venta en el PdV.

Video 1. "Vendiendo en el PdV"

Podemos venderlo porque aparece marcado en la pestaña Ventas (antes Punto de Venta), "Disponible en TPV":

A screenshot of a POS software interface. At the top, there's a navigation bar with icons for Punto de venta, Tablero, Pedidos, Productos, Informes, and Configuración. Below the navigation bar, there's a toolbar with buttons for Nuevo, Productos, Extra Prices, Documentos, A Mano, and Más. In the main area, there's a form for a product named "Desk Pad" with a star icon. Underneath the name, there are two checked checkboxes: "Puede ser vendido" and "Puede ser comprado". Below the checkboxes, there are tabs for Información General, Ventas, Compra, Inventario, and Contabilidad. The "Ventas" tab is selected. At the bottom of the screen, there are two sections: "AUMENTAR VENTAS Y CROSS-SELL" and "PUNTO DE VENTA". In the "AUMENTAR VENTAS Y CROSS-SELL" section, there's a link for "Productos opcionales" and a note about recommending items to the cart or quote. In the "PUNTO DE VENTA" section, there's a checkbox for "Disponible en TPV" which is checked, and another checkbox for "Para pesar con" which is unchecked.

Para realizar la venta iniciaremos sesión en el módulo Punto de venta

The screenshot shows the POS software interface. On the left, there's a shopping cart icon with the text "This order is empty". Below it is a table with columns for "Reembolso" (Refund), "Nota del Cliente" (Customer Note), and "Cotización/Orden" (Quotation/Order). A sidebar on the left has sections for "Cliente" (Client) and "Pago" (Payment). On the right, there's a search bar and a grid of products:

Reembolso	Nota del Cliente	Cotización/Orden		
Cliente	1	2	3	Qty
Pago	4	5	6	% Disc
	7	8	9	Price
	+/-	0	,	☒

Product Grid:

- Acoustic Block Screens: 423,50 €
- Desk Organizer: 24,20 €
- Desk Pad: 42,35 €
- Drawer: 108,90 €
- Flipover: 96,80 €
- LED Lamp: 242,00 €

Aquí veremos todos los productos que tenemos en venta. Si queremos vender “Desk Pad”, pulsamos sobre él. Podremos realizar descuento sobre el producto pulsando “Desc” (% Disc en inglés) y se agregará el descuento. Para agregar otro producto sin descuento pulsaremos sobre “Cant” (“Qty” en inglés) y este ya no tendrá descuento. Al realizar el pago tendrá la opción de Efectivo o Banco (Cash o Bank, aunque también aparece en la versión de dic/2023 la opción “Customer Account”). Si el pago es efectivo (“Cash”) nos pedirá que introduzcamos la cantidad recibida por el cliente y automáticamente nos muestra el cambio a entregar de vuelta al cliente, y nos da oportunidad de imprimir el recibo.

80,47 €

Pago satisfactorio

[Imprimir recibo](#)

Email: el recibo, la factura [Email](#)

	101
Desk Pad	38,12 €
1,00 Unidades x 38,12 € / Unidades	
Con un 10% descuento	
Desk Pad	42,35 €
1,00 Unidades x 42,35 € / Unidades	
TOTAL	80,47 €
Cash	100,00
CAMBIO	19,53 €
Descuentos %	4,24 €
Número de identificación fiscal	ExVAT
IVA 21 %	Total
	13,97 66,50 80,47
%	
Pedido 00001-001-0001	
26/12/2023 11:43:02	

> Nuevo pedido

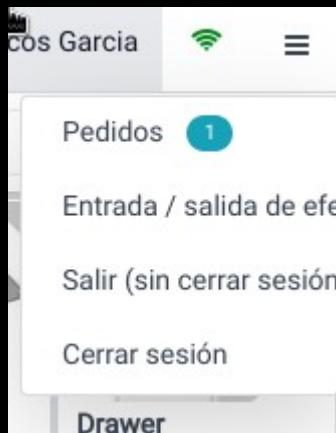
Al cerrar la sesión de punto de venta podremos comprobar que del producto Desk Pad ahora disponemos de 48 unidades, dos menos que son las que hemos vendido.



Cerrando la sesión en el PdV

Al finalizar el horario de la tienda, cuando vayamos a cerrar, es necesario cerrar el PdV. Para ello, basta con pulsar sobre “Close” (“Cerrar sesión” en la versión de dic/2023) en la esquina superior derecha.

Botón “Cerrar sesión” en versión de dic/2023



Botón “Close” en otra versión anterior



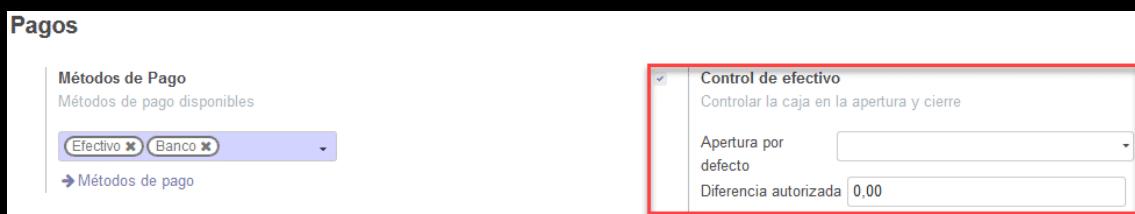
Una vez confirmado, debemos cerrar el PdV por completo: Cerrando el PdV mediante “Cerrar”.

Y validar y contabilizar los asientos de cierre: Validando y contabilizando los asientos de cierre.

Todo es intuitivo, luego nos pedirá confirmación del cierre de sesión...

Con Odoo es posible seguir vendiendo en nuestro PdV, incluso si se ha perdido la conexión a internet. El sistema almacena en memoria las ventas y estas se sincronizan una vez que la conexión vuelve.

De esta forma, aún no hemos llevado un **control del dinero en efectivo en la caja**. Para poder hacer esto, debemos **activar el checkbox de Control de efectivo** de la **configuración del PdV**. Esto nos ayuda a controlar la caja en la apertura y cierre del PdV. (En la versión de dic/2023 parece que esta opción no aparece, de modo que supuestamente lo hace de forma predeterminada).



La devolución de un producto en el PdV

La devolución se puede hacer en el **backend**, pero **no es lo recomendable**. De hecho, un usuario final “convencional”, normalmente, **no sabe cómo hacerlo**, ya que no se le forma para ello. Por esto, **lo hacemos en el frontend** (la interfaz del PdV).

Además, en algunos países no es del todo legal hacerlo en el backend. Para realizar la devolución de uno de los productos comprados por el cliente, tenemos que indicar la **cantidad como -1** en el PdV y pulsar sobre “**Pagos**” (en la versión actual pulsaremos la tecla “1” y se restará “-1” y pulsamos sobre “Reembolso”).

Finalmente, como el pago se realizó en efectivo, le **devolvemos el efectivo**, generándonos un ticket con un **cambio de 38,12€**.

The screenshot shows a POS system interface with the following details:

- Header:** Includes a "Volver" button, a "Nuevo pedido" button, a search bar ("Buscar pedidos..."), and a dropdown menu set to "Pagado".
- Product Selection Area:** A modal window titled "Seleccione el producto o producto(s) a reembolsar y establezca la cantidad" (Select the product or products to be refunded and set the quantity). It lists a "Desk Pad" item with a quantity of "-1,00" and a unit price of "38,12 € / Unidades". The total amount shown is "Total: -38,12 €" and includes a note about taxes: "Impuestos: -6,62 €".
- Order Table:** A table showing two orders:
 - Order 1:** Date 26/12/2023 12:41:01, Number 00003-001-0001, Status Pagado. Details: Simplified Invoice Partner (ES), Client Marcos Garcia, Total -38,12 €.
 - Order 2:** Date 26/12/2023 11:42:57, Number 301, Status Pagado. Details: Simplified Invoice Partner (ES), Client Marcos Garcia, Total 80,47 €.
- Action Buttons:** Buttons for "Reimprimir factura" (Print receipt) and "Imprimir recibo" (Print receipt).
- Refund Area:** A large button labeled "Reembolso" with a right-pointing arrow. To its left is a placeholder for a QR code.
- Quantity Input:** A numeric keypad with buttons for 1, 2, 3, % Disc, 4, 5, 6, Price, ., and +/-.

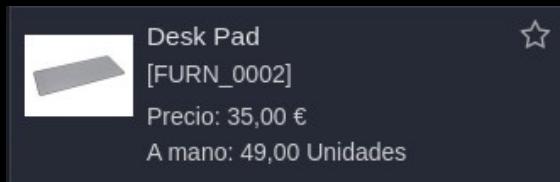
Desk Pad -38,12 €
-1,00 Unidades x 38,12 € / Unidades
Con un 10% descuento

TOTAL -38,12 €

Cash -38,12
CAMBIO 0,00 €
Descuentos -4,24 €
% Número de identificación ExVAT Total
de fiscal
IVA
21 -6,62-31,50-38,12
%

Pedido 00003-001-0001
26/12/2023 12:41:05

Ahora, volvemos a tener 49 unidades del producto, en lugar de las 48 que teníamos.



La fabricación propia

Ventajas de la fabricación propia.

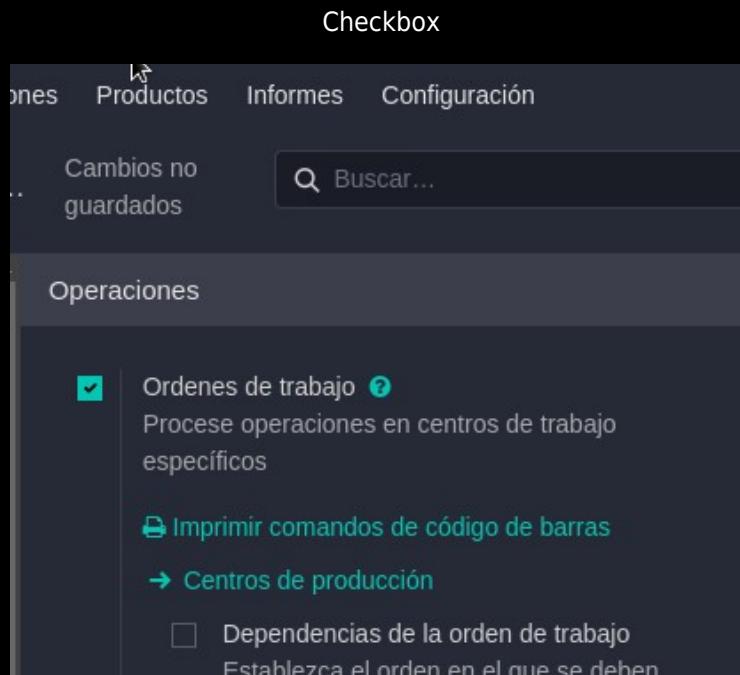
La fabricación propia en nuestra empresa de muebles.

Continuando con el ejemplo de nuestra empresa de muebles, ahora hemos decidimos que, además de importar muebles al por mayor desde otras partes del mundo, vamos a fabricar nuestros **propios muebles**. Esto mejorará nuestros **márgenes de beneficio** y nos permitirá disponer de los **productos más rápidamente** y con la **calidad** que nosotros deseamos. Además, podremos vender nuestros productos con el **distintivo** de '**fabricado en España**'.

Para poder fabricar nuestros propios productos, hacemos uso del **módulo de Fabricación**.

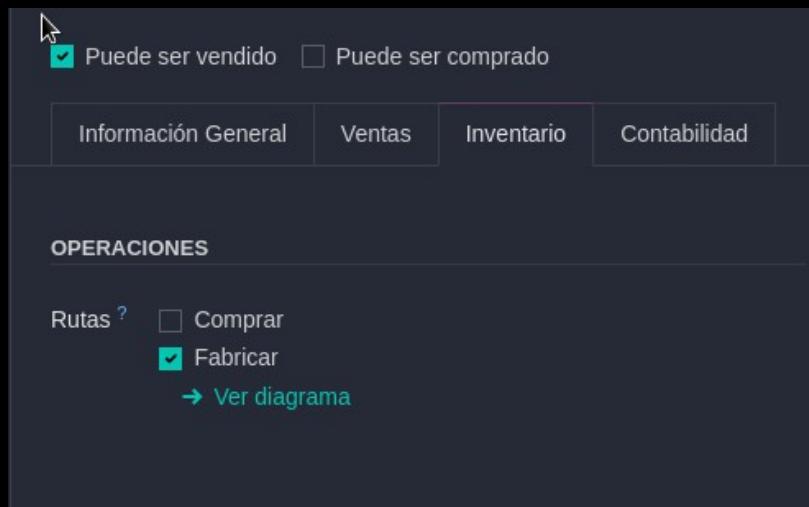
Este módulo **requiere** tener instalados también los de **Contabilidad e Inventory**, pero estos ya los tenemos instalados del tema anterior. Lo **ideal** es poder contar también con el **módulo de Calidad** para poder agregar controles de calidad a los productos y chequear que todo se fabrica correctamente. Pero este está disponible en la **versión Enterprise**, por lo que solo haremos uso del módulo de Fabricación.

Para poder procesar operaciones en nuestra fábrica basadas en rutas de producción, activaremos el **checkbox Órdenes de trabajo** en los ajustes del módulo de Fabricación.



Definiendo la lista de materiales

Vamos a “**Productos**” en el **módulo de “Fabricación”** y creamos nuevo producto. Podemos llamarlo “Mesa”. Desmarcamos “puede ser vendido” ya que lo fabricaremos nosotros. Será “**Almacenable**” y le pondremos un precio de venta (al cliente) y un coste (lo que nos costará fabricarlo). En la **pestaña “Inventario”** desmarcamos “Comprar” y **dejaremos “Fabricar”**.



Luego, desde “**lista de materiales**” tendremos que crear uno nuevo, agregando los componentes con productos ya adquiridos con anterioridad en nuestra lista de productos:

- “superficie de mesa”,
- “pata de mesa”,
- “tornillo”.

Fabricando el producto

Tras la **creación de la lista de materiales necesarias** para la fabricación del producto, hemos recibido nuestro primer encargo, 3 mesas. Para ello, creamos una nueva orden de producción:

Creación de una orden de producción



A continuación, pulsaremos en “**Crear**” (“Nuevo”) y al introducir en el producto la “**Mesa**” y en **cantidades “3”**, se emite el informe con los **materiales necesarios**, que en este caso son 3 superficies, 12 patas y 12 tornillos:

Materiales necesarios para la fabricación

Producto	Cantidad	Fecha prevista	Fin	Responsable
Mesa	3,00	26/12/2023 13:26:44	26/12/2023 14:26:44	Marcos Garcia

Componentes	Ordenes de trabajo	Varios
superficie de mesa		
Pata de mesa		
Tornillo		

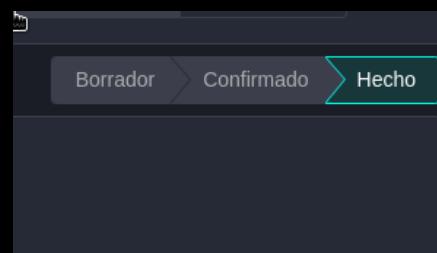
Seguidamente, pulsaremos sobre el botón de “**Marcar por realizar**” (“Confirmar”) y esto hace que la orden de producción pase del estado “borrador” a “**confirmado**”. Ahora, si pulsamos sobre “**Comprobar disponibilidad**”, podemos apreciar si tenemos disponibilidad o no de materiales (productos para la fabricación). El material disponible aparecerá en color verde.

Cuando el **stock para fabricar** (módulo de Fabricación) un producto es **insuficiente**, tenemos que **realizar una compra** (módulo de **Compras**) que nos ha permitido **aumentar el stock** (módulo de **Inventario**) para **poder fabricar el producto** que después podremos vender en nuestras tiendas (módulo de Punto de venta).

Tras realizar un pedido de compra para suplirnos de dichos materiales (si no los tenemos en el almacén), tendremos todos en verde.

Componente	A consumir	Cantidad
superficie de mesa	3,00	3,00
Pata de mesa	12,00	12,00
Tornillo	12,00	12,00

Ahora ya podemos continuar con la **fabricación**. Para ello, pulsamos sobre el botón “**Producir**” (“producir todo”) y su estado pasa a “**Para cerrar**” (“confirmado” en la última versión). En el momento en el que terminen de fabricarse y pulsemos en “**Marcar como hecho**”, su estado pasa a “**Hecho**” y ya tenemos 3 unidades en stock de la mesa. (En la última versión pasa a “Hecho” directamente al “**Producir todo**”).



Stock actualizado del producto fabricado.



Configuración de rutas y centros de producción

Para la fabricación de la mesa, hemos organizado una línea de fabricación con **dos centros de producción**:

- Assembly Line 1.
- Paint Line 1.

	Centro de p...	Código	Etiqueta	Centros de trab...	Coste por h...	Capacidad	Tiempo de ...	OEE Objetivo
	Assembly Line 1				10,00	1,00	100,00	90,00
	Paint Line 1				10,00	1,00	100,00	90,00

Para que todo **fluya correctamente entre los centros** de producción, creamos una “**ruta**”, que denominaremos “**fabricación de mesa**”, con **dos operaciones** (líneas) “ensamblar” y “pintar”, ambas con una **duración de 60 minutos**. Además, la persona encargada de diseñar las mesas nos ha enviado un **documento PDF** con los **pasos** para ayudar a los trabajadores del **centro de producción “ensamblaje”**.

Para ello, debemos abrir el módulo de Fabricación, “**Datos principales**” (“información general en la última versión), pasar a “Productos/**Lista de materiales**” y ahí, en lugar de crear una lista de materiales como hicimos anteriormente, esta vez **editamos la lista “Mesa”**, concretamente, su “**ruta de producción**” (Pestaña “operaciones” en la última versión”).

	Producto	Referencia	Tipo de LdM
	Mesa	Mesa (nuevo) 1	Fabricar este producto

"Operaciones" en última versión dic/2023

Producto	Mesa	Referencia	Mesa (nuevo) 1	
Cantidad ?	3,00	Tipo de LdM		
		<input checked="" type="radio"/> Fabricar este producto	<input type="radio"/> Kit	
Componentes		Operaciones	Varios	
Operación	Centro de pro...	Cálculo de dur...	Duración (minutos)	Instrucc...
Agregar línea Copiar operaciones existentes				

Ruta de producción en una versión anterior

Lista de materiales / Mesa

[Guardar](#) [Descartar](#)

Producto	Mesa	Relacionado con:
Variantes de producto		Tipos de fabricación:
Cantidad	1,00	Costo:
Ruta de producción		
Componentes Varios		
Componente	Cantidad	Aplicación
[FURN_8522] Superficie de la Mesa	1,000	
[ELUDN_2232] Reta de mesa	4,000	

Creamos la ruta “fabricación de mesa” con **dos operaciones**:

- **Ensamblado**: en el centro de producción Assembly Line 1, con una duración de 60 minutos y adjuntando el PDF con las instrucciones.
- **Pintura**: en el centro de producción Paint Line 1, con una duración de 60 minutos.

Operación de ensamblado

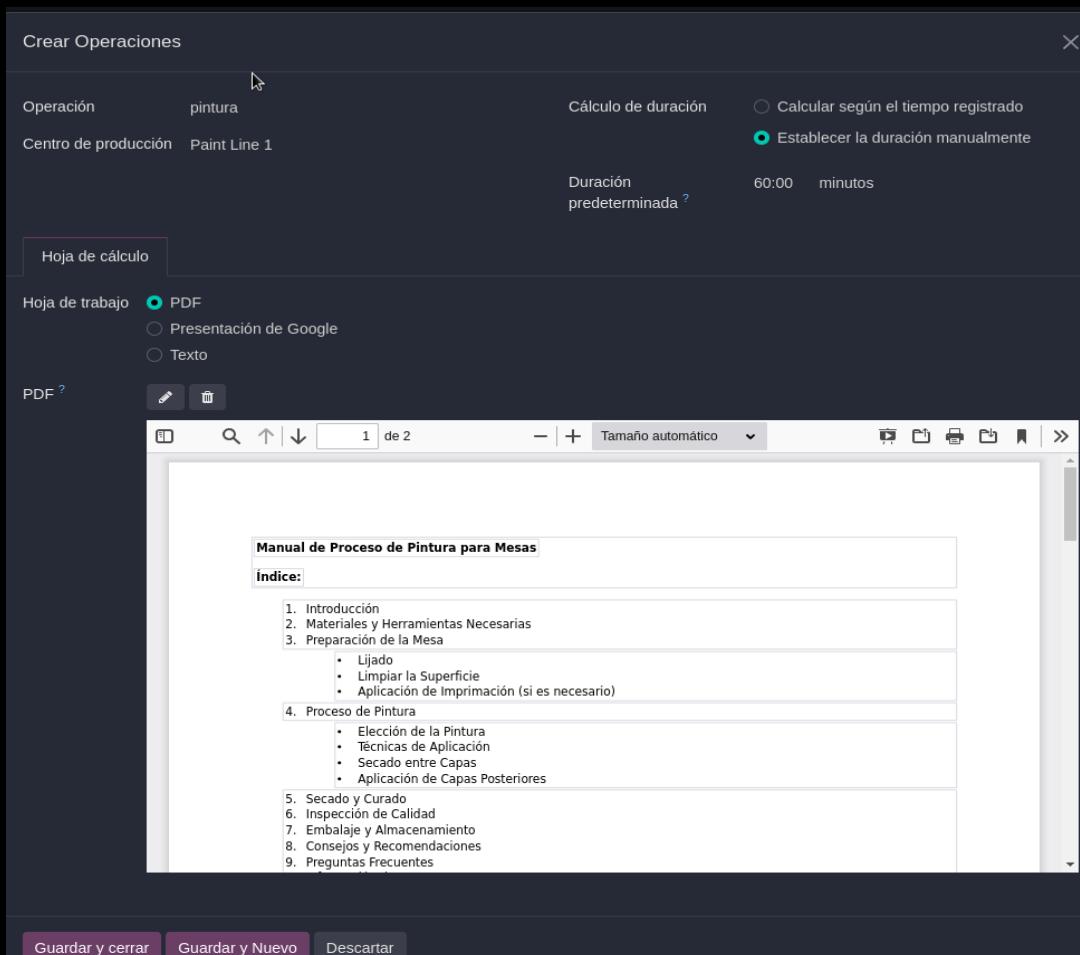
CrearOperaciones

Operación	ensamblado	Cálculo de duración	<input type="radio"/> Calcular basado en tiempo real <input checked="" type="radio"/> Establecer la duración manualmente
Centro de producción	Assembly Line 1	Duración predeterminada	60:00 minutos
Compañía	My Company (San Francisco)		
Comenzar siguiente operación	<input checked="" type="radio"/> Una vez procesadas todas las unidades <input type="radio"/> Una vez que algunos productos sean procesados		
Descripción Hoja de cálculo			
Hoja de cálculo	<input checked="" type="radio"/> PDF <input type="radio"/> Presentación de Google		
PDF	 		

En la última versión es “Crear Operaciones”

Crear Operaciones

Operación	Ensamblado	Cálculo de duración	<input type="radio"/> Calcular según el tiempo registrado <input checked="" type="radio"/> Establecer la duración manualmente
Centro de producción	Assembly Line 1	Duración predeterminada ?	60:00 minutos
Hoja de cálculo			
Hoja de trabajo	<input checked="" type="radio"/> PDF <input type="radio"/> Presentación de Google <input type="radio"/> Texto		
PDF ?	Suba su archivo		
Guardar y cerrar Guardar y Nuevo Descartar			



Quedando, por tanto, la ruta con dos operaciones y un total de 120 minutos, como se muestra en la siguiente imagen:

Ruta de producción configurada

Nuevo Lista de materiales Mesa (nuevo) 1: Mesa Rendimiento de las operaciones Vista general de LdM 1 / 1

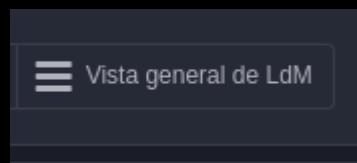
Producto	Mesa	Referencia	Mesa (nuevo) 1												
Cantidad ?	1,00	Tipo de LdM	<input checked="" type="radio"/> Fabricar este producto <input type="radio"/> Kit												
<table border="1"><thead><tr><th>Componentes</th><th>Operaciones</th><th>Varios</th></tr></thead><tbody><tr><td>Operación ^</td><td>Centro de producción</td><td>Cálculo de duración</td></tr><tr><td>Ensamblado</td><td>Assembly Line 1</td><td>Establecer la duración manualmente</td></tr><tr><td>pintura</td><td>Paint Line 1</td><td>Establecer la duración manualmente</td></tr></tbody></table>				Componentes	Operaciones	Varios	Operación ^	Centro de producción	Cálculo de duración	Ensamblado	Assembly Line 1	Establecer la duración manualmente	pintura	Paint Line 1	Establecer la duración manualmente
Componentes	Operaciones	Varios													
Operación ^	Centro de producción	Cálculo de duración													
Ensamblado	Assembly Line 1	Establecer la duración manualmente													
pintura	Paint Line 1	Establecer la duración manualmente													
Duración (minutos)															
0															
120:00															
Agregar línea Copiar operaciones existentes															

Controlando los costes

Podemos crear un **informe con el control de los costes** de la mesa que estamos fabricando, incluyendo **tiempo de mano de obra** (servicio) y **materiales**.

Para ello, desde el **módulo de Fabricación** de Odoo, debemos entrar en la “**Lista de materiales**” y, concretamente, en la lista de materiales del **producto “Mesa”**.

En la parte superior derecha nos encontramos con la pestaña de “**Estructura & Costo**” (En la última versión: “Vista general de LdM”).



Listado de estructura y costo de fabricación del producto “Mesa”

Mesa		3,00	1,00		
Referencia: Mesa (nuevo) 1		Disponible para usar	26/12/2023		
Producto	Cantidad	Plazo de entrega	Ruta	Coste de Lista de Materiales	Coste del Producto
Mesa	1,00	0 Días	Fabricar: Mesa (nuevo) 1: Mesa	15,00 €	115,00 €
Pata de mesa	4,00			0,00 €	40,00 €
superficie de mesa	1,00			15,00 €	15,00 €
Tornillo	4,00			0,00 €	4,00 €
▼ Operaciones	120:00			0,00 €	
Ensamblado - Assembly Line 1	60:00			0,00 €	
pintura - Paint Line 1	60:00			0,00 €	
		Costo unitario		15,00 €	115,00 €

Ahí aparece el listado que nos está pidiendo el contable de nuestra empresa, con el control de costes de la mesa, incluyendo materiales y tiempo de mano de obra.

Ese informe **podemos descargarlo y/o imprimirlo** para enviarlo al contable que nos lo ha pedido.

Resumen de la LdM					
Mesa					
Referencia: Mesa (nuevo) 1					
Producto	Cantidad	Plazo de entrega	Ruta	Coste de Lista de Materiales	Coste del Producto
Mesa	1,00	0 Días	Fabricar: Mesa (nuevo) 1: Mesa	15,00 €	115,00 €
Pata de mesa	4,00			0,00 €	40,00 €
superficie de mesa	1,00			15,00 €	15,00 €
Tornillo	4,00			0,00 €	4,00 €
Operaciones	120:00			0,00 €	
Ensamblado - Assembly Line 1	60:00			0,00 €	
pintura - Paint Line 1	60:00			0,00 €	
Coste unitario				15,00 €	115,00 €

Planificar un pedido de fabricación

Teniendo lista la configuración de la ruta en la creación del producto “Mesa”, podemos planificar la creación de una mesa y ver cómo esta va pasando por los distintos centros de trabajo. Al **crear un producto “Mesa”** se autoinforma de la **“ruta de producción”** (“varios/Tipo de operación” en la versión de dic/2023, y aparece ya en “Ordenes de trabajo” el Ensamblado y Pintura de los dos centros de producción ya creados) con la **ruta “fabricación de mesa”**:

Ruta de producción autoinformada y no editable



En la última versión sería algo así

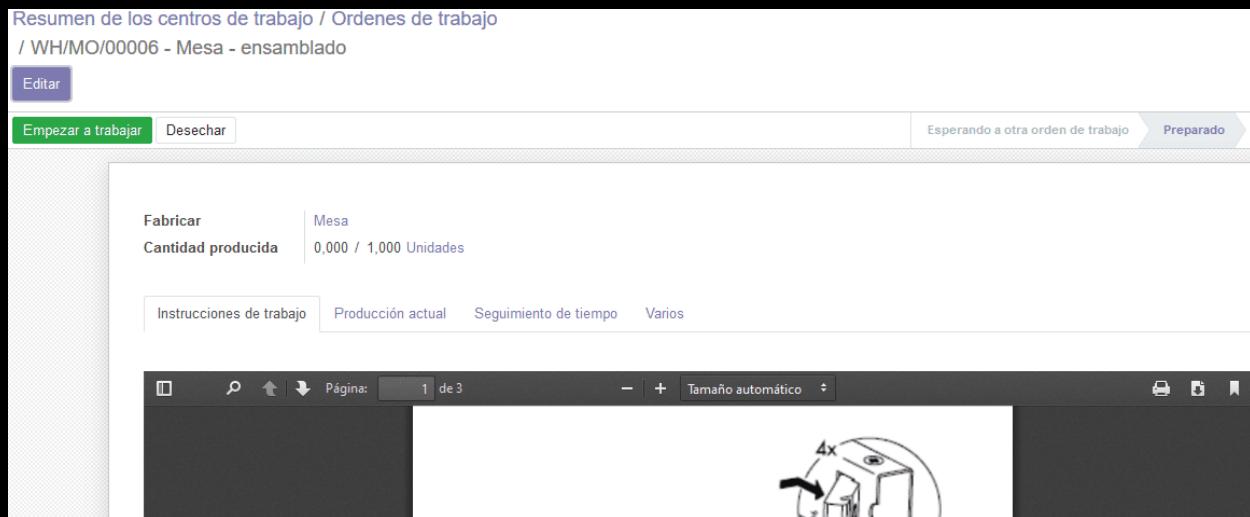
Operación	Centro de producción	Producto	Cantidad	Duración esperada	Duración real
Ensamblaje	Assembly Line 1	Mesa	3,00	60:00	<input checked="" type="checkbox"/>
Pintura	Paint Line 1	Mesa	3,00	00:00	<input checked="" type="checkbox"/>
Agregar línea					
				60:00	00:00



Tras **comprobar la disponibilidad** de materiales y pulsar el **botón “Plan”**, el **estado** de la orden de trabajo pasa a **“Planificado”** (“confirmado” en versión de dic/2023) y, en ese momento, en el centro de producción Assembly Line 1, tendrán una nueva orden de trabajo “ensamblado”.

Ensamblado (versión de dic/2023)

Ensamblado (en una versión anterior)



Los trabajadores, cuando empiecen a trabajar, pulsan el botón de “**Empezar a trabajar**” (“iniciar” en la versión dic/2023) pasando el estado de la orden de trabajo a “**En proceso**” (“En progreso” en versión dic/2023), y cuando **finalizan** el trabajo pulsan “**Hecho**” (pasando a “**Terminado**”).

“Preparado” en el centro de producción Assembly Line 1

The screenshot shows a software interface for managing work orders. The top navigation bar includes "Fabricación", "Información general", "Operaciones", "Planificación", "Productos", "Informes", "Configuración", and a notification icon with a red dot. Below the navigation, there is a search bar for "Ordenes de trabajo" (Work orders) with the result "1 seleccionado" (1 selected). There are buttons for "INICIAR" (Start), "DETENER" (Stop), "HECHO" (Done), "Imprimir" (Print), and "Acciones" (Actions). The main area displays a table of work orders:

Operación	Centro de ...	Producto	Cantidad	Duración ...	Duración r...	Estado	
<input checked="" type="checkbox"/> Ensamblado	Assembly Line 1	Mesa	3,00	240:00	00:00	Preparado	<button>Iniciar</button> <button>Bloquear</button> <input type="checkbox"/>
<input type="checkbox"/> Pintura	Paint Line 1	Mesa	3,00	180:00	00:00	Esperando a ...	<button>Iniciar</button> <button>Bloquear</button> <input type="checkbox"/>

“En progreso”

Ordenes de trabajo							Buscar...	1-2 / 2
	Operación	Centro de produc...	Producto	Cantidad	Duración estimada	Duración real	Estado	
<input type="checkbox"/>	Ensamblado	Assembly Line 1	Mesa	3,00	240:00	00:19	En progreso	<button>Detener</button> <button>Hecho</button> <button>Bloquear</button> <button>Marcado</button>
<input type="checkbox"/>	Pintura	Paint Line 1	Mesa	3,00	180:00	00:00	Esperando	<button>Iniciar</button> <button>Bloquear</button>
					420:00	00:00		

Órdenes de producción								Buscar...	1-1 / 1
	Referencia	Inicio	Producto	Actividad siguiente	Origen	Estado del componente	Cantidad	Estado	
<input type="checkbox"/>	WH/MO/00009	Hoy	Mesa	Preparación		Disponible	3,00	En progreso	
<input type="checkbox"/>									

En ese momento, en el centro de producción **Paint Line 1**, la orden de trabajo “pintura” que estaba en estado “Esperando a otra orden de trabajo” pasa a estado “**Preparado**”. Aquí los trabajadores deben seguir los mismos pasos para **terminar la orden de trabajo**.

Ordenes de trabajo							Buscar...	1-1 / 1
	Operación	Centro de produc...	Producto	Cantidad	Duración estimada	Duración real	Estado	
<input type="checkbox"/>	Pintura	Paint Line 1	Mesa	3,00	180:00	00:00	Preparado	<button>Iniciar</button> <button>Bloquear</button>

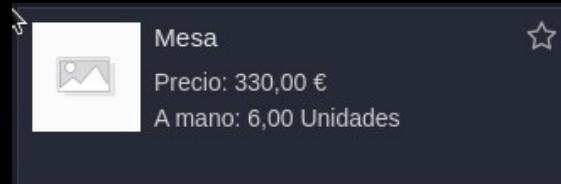
Ya solo falta **marcar como “Hecho”** la orden de producción y el stock de la mesa se incrementará en 1 unidad.

No planificado Desechar Desbloquear Desconstruir Imprimir etiquetas Borrador Confirmedo Hecho

Referencia de la orden de fabricación
☆ WH/MO/00009

Producto	Mesa	Fecha de inicio ?	26/12/2023 19:01:02
Cantidad	3,00 / 3,00	A producir	Fin ? 26/12/2023 19:08:59
Lista de materiales ? Mesa (nuevo) 1: Mesa		Responsable	M Marcos Garcia
Componentes	Ordenes de trabajo	Varios	
Producto	A consumir	Cantidad	
Tabla de mesa	3,00	3,00	
Pata de mesa	12,00	12,00	
Tornillo	12,00	12,00	

El Stock pasa de 3 a 6 unidades



Tal y como hemos visto en el transcurso del tema, Odoo puede ayudarnos en nuestras tiendas de venta de componentes informáticos y equipos de sobremesa. Concretamente, es muy útil hacer uso del módulo de Punto de venta, y del **módulo de Fabricación** (junto con los de **Contabilidad e Inventario**).

El módulo de **Punto de venta** nos permite gestionar la **venta de componentes** en las tiendas con acciones como la venta de productos, las devoluciones, el control del dinero de la caja, etc.

Por otro lado, el módulo de **Fabricación** nos permite **ir fabricando** equipos cuando necesitamos (bajo demanda, cuando el stock sea bajo, etc.).

Podemos, por ejemplo, crear **rutas** (tipo de operación) y **centros de trabajo** para diferenciar el departamento hardware y el software a la hora de fabricar los equipos, crear listas de materiales con los componentes de cada equipo, etc.

SISTEMAS DE GESTIÓN EMPRESARIAL

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Bases de datos con PgAdmin

Conexión a base de datos de Odoo

Tablas, vistas, campos y relaciones de tablas en Odoo

Informes y gráficos

Consultas SQL en PgAdmin

Modificación de valores desde PgAdmin

Backups y restauraciones

Vamos a ver una herramienta que nos permita gestionar la base de datos de Odoo, Open Source, con una interfaz amigable, pero a la vez potente. Nos permitirá realizar backups, ver informes de la base de datos, modificar datos directamente en la base de datos, etc.

La base de datos y PgAdmin

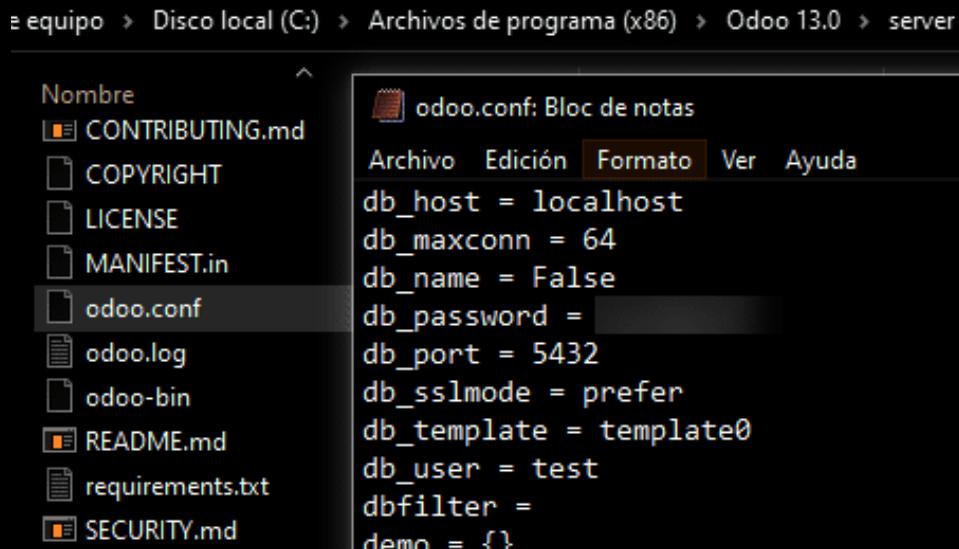
Odoo necesita un sistema de base de datos PostgreSQL para funcionar.

PostgreSQL es un potente sistema de base de datos objeto-relacional de código abierto que usa y amplía el lenguaje SQL.

Al instalar Odoo con el instalador en Windows, éste ya instalaba y configuraba PostgreSQL para su uso. Pero tenemos que aprender a configurar la conexión con la base de datos y saber cómo funciona, por si elegimos otro sistema operativo, otro modo de instalación, necesitamos crear roles, conectarnos a otra base de datos, hacer backups, por si surgen problemas, etc.

Para empezar, en el **archivo de configuración** de Odoo (**odoo.conf**) podemos ver como se está conectando actualmente Odoo a la BD (**host**, **usuario**, **contraseña**, **puerto**, etc.):

fichero odoo.conf



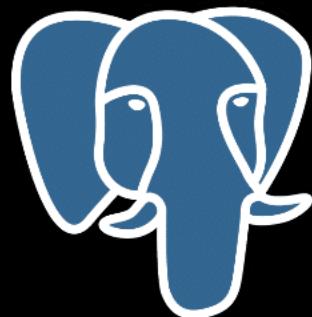
```
db_host = localhost
db_maxconn = 64
db_name = False
db_password =
db_port = 5432
db_sslmode = prefer
db_template = template0
db_user = test
dbfilter =
demo = {}
```

Para seguir conociendo como configurar PostgreSQL, lo haremos a través de la herramienta PgAdmin.

PgAdmin

PgAdmin es una aplicación muy popular que se usa tanto para la **gestión** como para la **administración** y el **desarrollo de bases de datos en PostgreSQL**.

Esta herramienta es también Open Source y multiplataforma. Probablemente se haya **instalado junto con PostgreSQL** (si instalamos en Windows), pero si no, bastará con descargarla e instalarla desde su web oficial. En nuestro caso usaremos la versión más actual (la 4). En enero de 2024 la actual es la 7.

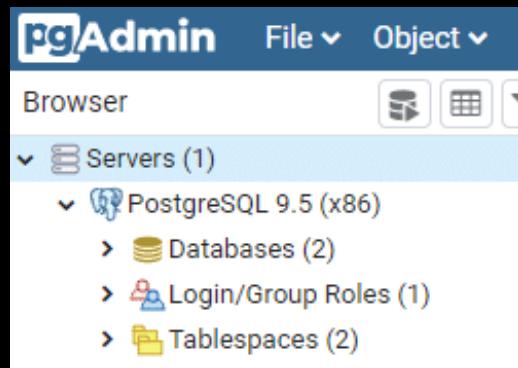


La conexión en PgAdmin

Al abrir PgAdmin por primera vez, si ya tenemos Odoo instalado, nos pedirá la contraseña para el usuario (en este caso db_user = test), que es la que vimos en la imagen del fichero odoo.conf mas arriba.

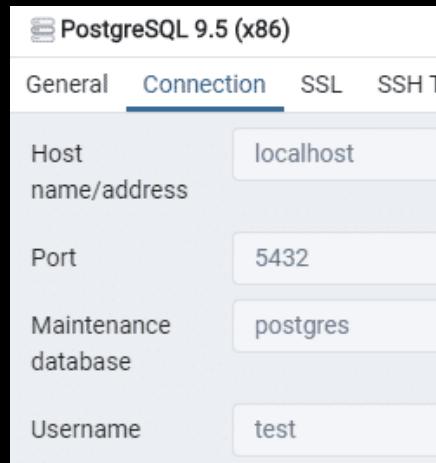
Una vez dentro, podremos apreciar que existe un servidor ‘PostgreSQL 9.5 (x86)’ (en enero 2024 el servidor será PostgreSQL 12), en cuyo interior hay **2 bases de datos**, un **usuario** de login y **2 tablespaces**:

Panel principal de PgAdmin



Si pulsamos con el **botón derecho** sobre el **servidor** ‘PostgreSQL 9.5’ y abrimos sus **propiedades**, podremos comprobar en la **pestaña de conexión** que se trata de la **misma configuración del fichero odoo.conf**:

Configuración de la conexión del servidor en PgAdmin



De esta forma comprobamos que **PgAdmin** está **conectado a la base de datos** con la que actualmente estamos trabajando **en Odoo**.

Al haber conectado **PgAdmin** con el servidor que usa Odoo, comentar que los cambios que se realicen sobre esta herramienta **afectarán directamente a Odoo**. Es decir, desde PgAdmin **podremos**, por ejemplo, hacer que un **campo** de una tabla sea obligatorio, **crear/modificar/eliminar registros** de una tabla, crear/modificar/eliminar **tablas**, etc. y todo esto tendrá la **misma repercusión** que si lo hiciésemos **directamente en Odoo**.

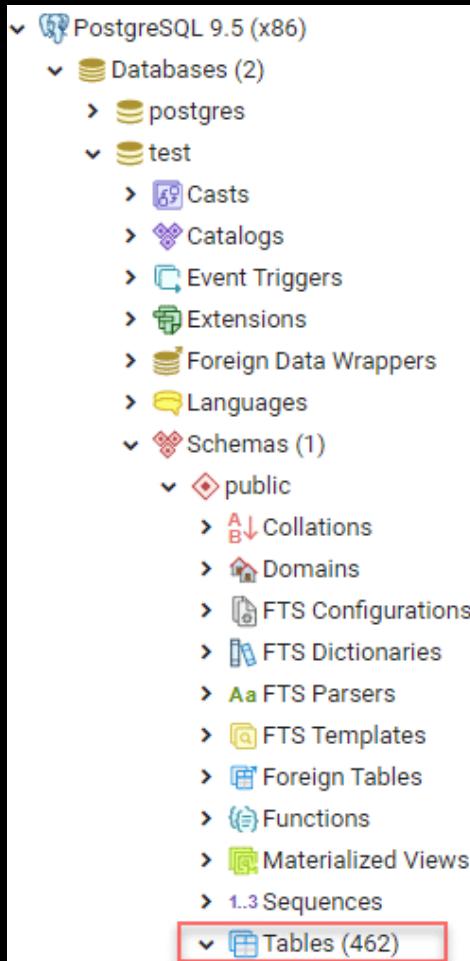
Tablas, vistas y campos de la base de datos

A continuación, vamos a ver las **tablas** y **las vistas** (con sus respectivos **campos**) que configuran la **base de datos de Odoo**.

Para **ver las tablas** en **PgAdmin** deberemos desplegar la siguiente **ruta del árbol**:

→ **servidor / base de datos / schemas / public / tables**.

Desde PgAdmin bastará con **desplegar el árbol** que se muestra en la siguiente imagen:



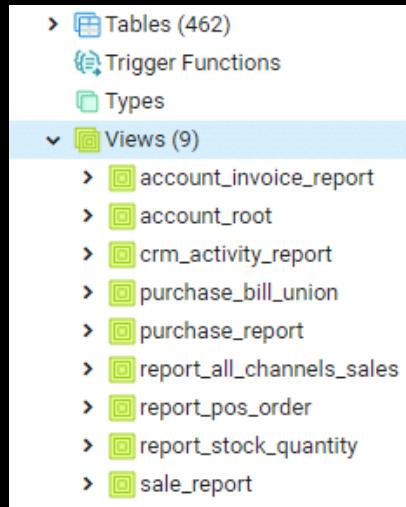
Podemos ver que disponemos de **462 tablas** (454 en la versión 17) en la base de datos.

Seguidamente, si desplegamos la **sección ‘tablas’**, podemos ver cada una de ellas con sus **características, propiedades, columnas**, etc.

Por ejemplo, en la tabla `product_template` nos encontramos con 45 columnas, 10 restricciones o constraints y 3 índices.

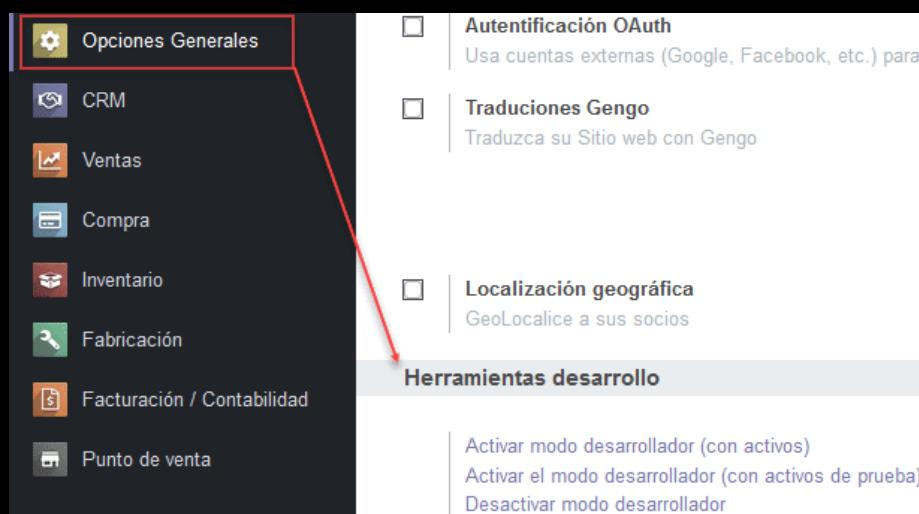
Al **seleccionar la tabla**, alguna de sus columnas o cualquier objeto, en la derecha, podremos ver sus **propiedades**, su **declaración SQL**, sus **estadísticas** (cuantas veces se ha leído, el tamaño de la tabla, cuantas veces se han insertado registros, etc.) y las **dependencias**.

Además de las tablas y sus campos también podremos analizar las '**vistas**' que se van **generando, junto con sus campos**.

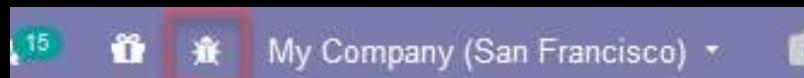


Tablas, vistas y campos en Odoo

Para poder visualizar información técnica en Odoo, deberemos entrar en el **modo desarrollador** de la herramienta. Para ello tendremos que entrar en las **opciones generales** de los ajustes de Odoo y pulsar sobre '**activar modo desarrollador (con activos)**'.



En ese momento, entre otras cosas, se nos habilitarán las herramientas '**Open Developer Tools**' (ícono en forma de bicho, igual al ícono "debug" en otros IDEs) y muchas opciones en el menú de configuración del módulo de ajustes.



Y en Ajustes tendremos una nueva pestaña llamada “Técnico”.

Viendo la base de datos desde Odoo

Una vez habilitado el modo desarrollador, ya podremos **visualizar las tablas, vistas y campos en Odoo**.

En esta pestaña “Técnico”, encontraremos “Estructura de la base de datos”:

- “Precisión decimal”, donde podremos modificar el número de decimales de un campo.
- “Modelos”, donde podremos ver:
 - los campos de cada tabla,
 - los permisos de acceso (de otras tablas para actualizar, eliminar sobre esta tabla),
 - reglas de registro,
 - notas,
 - vistas,
 - en qué módulos (aplicaciones) de Odoo se usa la tabla, es decir, qué módulos tienen acceso a dicha tabla;
- “Campos”, donde vemos todos los campos de todas las tablas. Buscamos el campo y a la derecha vemos el modelo (tabla) en el que se usa, y si pulsamos en el campo, vemos todas sus características:



The screenshot shows a table with the following columns: Nombre de campo (Field Name), Etiqueta de campo (Field Label), Modelo (Model), Tipo de campo (Field Type), Tipo (Type), Indexado (Indexed), Almacenado (Stored), Sólo lectura (Read-only), and Relación del objeto (Object Relation). The table lists several fields for the 'product' model, such as 'account_src_id', 'attribute_line_ids', 'bom_product_template_attribute_ids', 'byproduct_id', 'byproduct_ids', 'categ_id', and 'cateo_id'. Each row includes a checkbox for selection and a detailed description of the field's properties.

Nombre de campo	Etiqueta de campo	Modelo	Tipo de campo	Tipo	Indexado	Almacenado	Sólo lectura	Relación del objeto
account_src_id	Cuenta de producto	Asignación de cuentas de Posici...	many2one	Campo ba...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	account.account
attribute_line_ids	Atributos del producto	Producto	one2many	Campo ba...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	product.template.attribute.line
attribute_line_ids	Atributos del producto	Plantilla de producto	one2many	Campo ba...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	product.template.attribute.line
bom_product_template_attribute_ids	Aplicar en variantes	Línea de Lista de Materiales	many2many	Campo ba...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	product.template.attribute.val...
byproduct_id	Subproductos	Movimiento de existencias	many2one	Campo ba...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	mrp.bom.byproduct
byproduct_ids	Subproductos	Lista de materiales	one2many	Campo ba...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	mrp.bom.byproduct
categ_id	Categoría de producto	Capa de valoración de stock	many2one	Campo ba...	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	product.category
cateo_id	Categoría de producto	Línea inventario	many2one	Campo ba...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	product.category

- “Selección de campos (Fields selection)”, donde vemos las opciones de cada campo, por ejemplo si un producto es consumible, servicio o almacenable, y además se pueden volver a configurar desde aquí.
- “Restricciones del modelo”, por ejemplo la “f” nos indica que es una clave foránea, y con qué módulos se relacionan estas restricciones.
- “Relaciones Many to Many”.
- “Adjuntos” de imágenes, facturas... que también se almacenarán en una base de datos. Y podremos abrir las y ver su contenido, ya que es un adjunto generado anteriormente.

Estas opciones y herramientas ayudarán a los desarrolladores a **trabajar con Odoo**. Si vamos a **trabajar técnica o funcionalmente con Odoo**, deberemos familiarizarnos con el **modo desarrollador**.

Extraer informes y gráficos sobre la actividad del servidor en tiempo real

Desde la versión 4 de PgAdmin es posible realizar una **monitorización en tiempo real** del **estado del servidor** de la base de datos haciendo uso del **Dashboard**. Para ello deberemos posicionarnos sobre la base de datos a analizar y pulsar sobre Dashboard. En ese momento aparecerán una serie de gráficas:

- La grafica “**Data sessions**”:

muestra las sesiones de la base de datos:

en verde las que **están activas**,
en rojo las **desactivadas**,
en azul las **totales**.

- La grafica “**transacciones por segundo**”:

en azul las **transacciones**,
en verde los **commits** (transacción para guardar),
en rojo los **rollbacks** (transacción para deshacer).

- La grafica “**Tuples in**”:

muestra las acciones de **modificaciones** sobre la base de datos:
en azul las **inserciones**,
en verde las **actualizaciones** ,
en rojo las **eliminaciones**.

- La grafica “**Tuples out**”:

muestra las lecturas a la base de datos:
en azul las **lecturas**,
en verde las **devoluciones de datos**.

- La grafica “**Block I/O**”:

muestra los bloques de entrada/salida:
en azul los de **salida**,
en verde los de **entrada**.

Además, en la parte inferior aparece información sobre la **actividad del servidor**, pudiendo incluso **cerrar sesiones**, **cancelar querys**, etc.

Actividad del servidor

Server activity								 Search	
	Sessions	Locks	Prepared Transactions						
	PID	User	Application	Client	Backend start			State	Waiting?
	3660	test		::1	2020-07-10 18:58:52 CEST			idle	no
	3996	test		::1	2020-07-10 18:58:56 CEST			idle	no
	5000	test		::1	2020-07-10 18:58:55 CEST			idle	no

Usos de PgAdmin:
Conexión Odoo y BBDD

A continuación, veremos algunos usos que le podremos dar a **PgAdmin** y cómo nos ayudará con la **gestión de la BD de Odoo**.

Importante:

Otra de las opciones que hemos habilitado con el **modo desarrollador** es la ayuda que Odoo nos proporciona con la **información de los campos**. Ahora, al posicionarnos durante 1 o 2 segundos sobre, por ejemplo, un campo de un formulario, nos aparecerá un pop-up con información referente a dicho campo. Por ejemplo, si nos vamos al producto 'Mesa' que fabricamos en temas anteriores (módulo de fabricación / datos principales / productos / mesa) y posicionamos el cursor sobre el campo 'Precio de venta', Odoo nos indicará que se trata del campo 'list_price' del **modelo 'product.template'** (en PgAdmin será la **tabla 'product_template'**), que es de tipo 'float' y que se muestra como 'Monetary' (moneda):

Pop-up con información técnica del campo



Pero dicha tabla puede tener muchos registros. Para saber concretamente qué registro es concretamente la 'Mesa' cuyo valor 'list_price' es de 330€ podemos **buscar el 'id'** de varias formas:

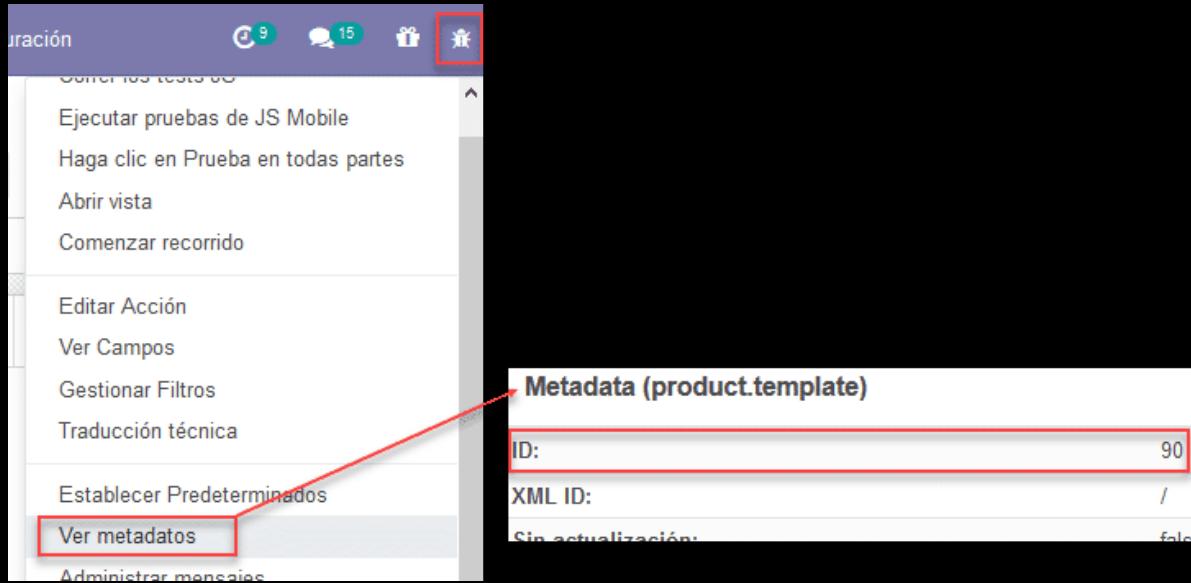
- En la **barra de dirección** (donde además podremos ver la tabla como "model="):

Viendo el 'id' del producto en la barra de direcciones

```
localhost:8069/web?id=90&action=492&model=product.template&view_type=form&cids=&menu_id=313
```

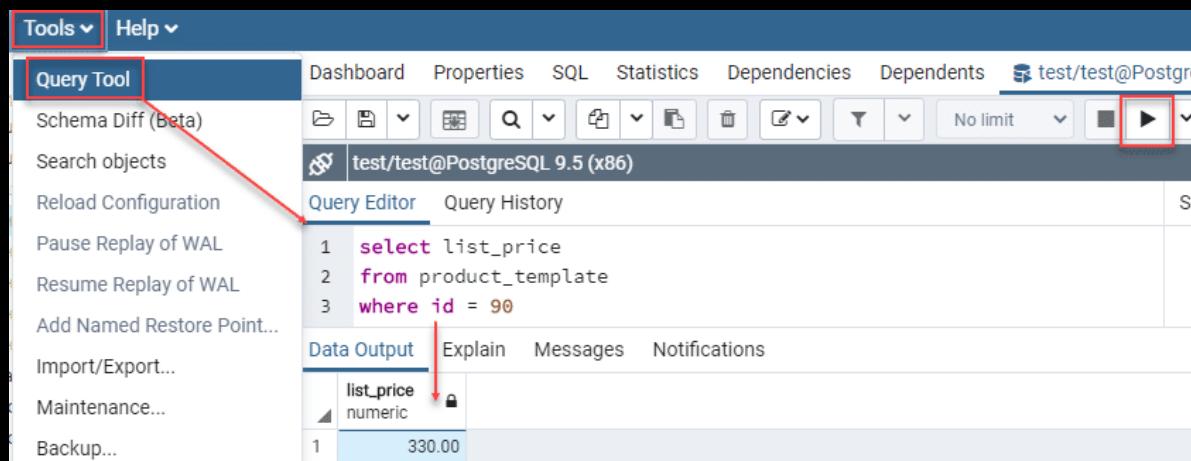
- O accediendo a los **metadatos** (pulsando el icono de **modo desarrollador** y yendo a "Ver metadatos"):

Viendo el 'id' del producto en los metadatos



Así, con la tabla (product_template), el id (90) del registro y el nombre del campo (list_price), podríamos realizar una **consulta de acceso a datos en PgAdmin** para buscar dicho valor (330.00):

Consulta de acceso a datos desde PgAdmin



Usos de Pgadmin

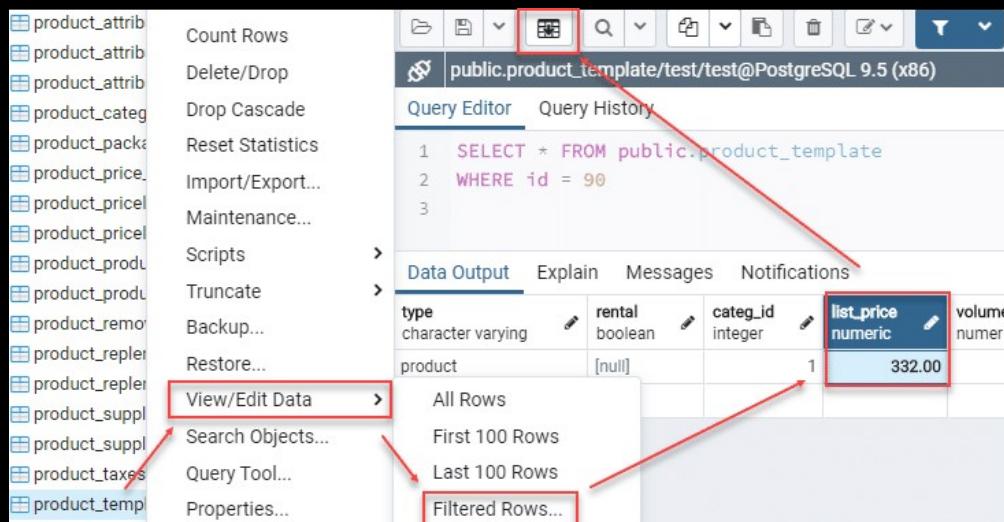
Modificar valores

Los **cambios** que se realicen **en PgAdmin afectarán directamente a Odoo**. Aunque lo más fácil y recomendable sería hacerlo desde Odoo, hay veces que nos ayudará realizar la modificación directamente en la base de datos, por ejemplo, para **modificaciones masivas**.

Así, por ejemplo, si quisiéramos **modificar el valor del precio** del producto 'Mesa' **desde PgAdmin** podríamos hacerlo de varias formas:

1. Actualizando el valor desde la tabla en PgAdmin (filtrando por id=90):

Seleccionando la tabla, botón derecho se selecciona "View/Edit Data" → "Filtered Rows", y en el cuadro escribimos el filtro, en este caso: id = 90. Damos a "ok" y nos abre el resultado, ejecutando la sentencia completa pgAdmin, como se ve en la imagen. Y ya podemos buscar en la pestaña "Data Output" el campo "list_price", que se podrá modificar pulsando sobre el cuadro de texto del precio actual. Y guardamos con el icono "Save data changes".



Podemos ver cómo en Odoo los cambios se ven reflejados al instante.

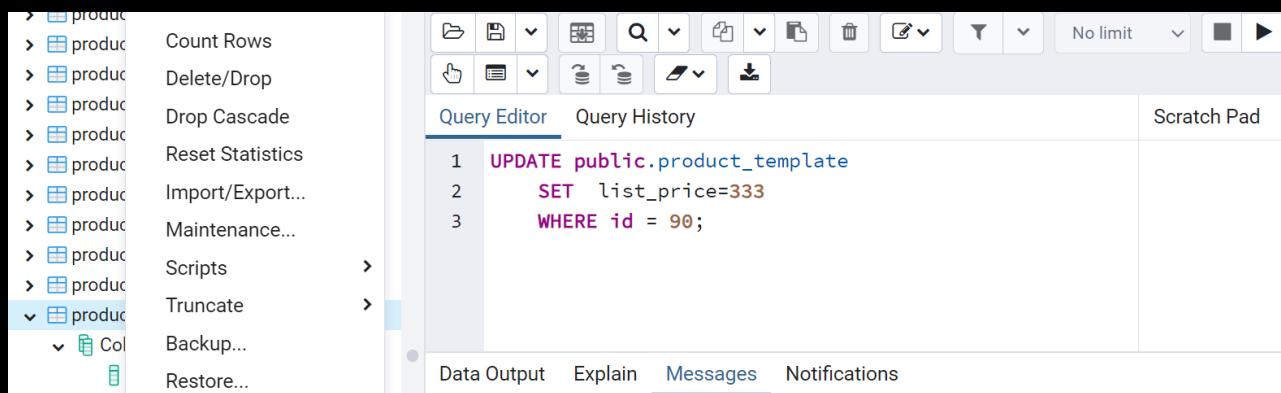
Comprobando en Odoo que se ha modificado el valor



2. Actualizando el valor mediante código SQL:

De la misma manera, pero “Scripts” → “UPDATE Script”, y modificando la sentencia SQL:

Actualizando un valor con una sentencia SQL en PgAdmin.



Si ahora, para volver al valor que tenía (330€), lo **modificamos desde Odoo**, podemos ver en **PgAdmin que éste se actualiza**:

Comprobar que ha cambiado en PgAdmin

```
Query Editor Query History

1 SELECT list_price
2 FROM public.product_template
3 WHERE id=90

Data Output Explain Messages Notifications

list_price
numeric
1 330.00
```

Las relaciones

En Odoo pueden darse relaciones:

many2one: tipo de campo que almacena una relación de **muchos (n) a uno (1)**,

one2Many: (tipo de campo que almacena una relación de **uno (1) a muchos (m)**),

many2many: (tipo de campo que almacena una relación de **muchos (m) a muchos (n)**).

Se pueden ver esas **relaciones en Odoo**:

Para ello es necesario entrar en el modo desarrollador. Por ejemplo, una relación one2many podría ser la de un **pedido de ventas** (one) y todos sus productos (many). Si vamos al módulo “Ventas” y elegimos un presupuesto, al entrar en **modo desarrollador** y posicionarnos sobre el **listado de productos** (etiqueta Producto sobre la lista), el **pop-up nos indica que es un tipo one2many de la tabla** (objeto-modojo) **sale.order** con la **tabla** (relación) **sale.order.line**.

The screenshot shows the Odoo Sales module interface. At the top, there's a purple header bar with tabs: Ventas, Pedidos, A facturar, Productos, Informes, and Configuración. Below the header, a blue banner displays "Presupuestos / S00007". On the left, there are two buttons: Editar and Crear. To the right of these buttons are two dropdown menus: Imprimir and Acción. The main content area has a large title "S00007". Below the title, there are two sections: "Cliente" and "Plantilla de presupuesto". The "Cliente" section contains the address: Gemini Furniture, 317 Fairchild Dr, Fairfield CA 94535, Estados Unidos. The "Plantilla de presupuesto" section is empty. At the bottom of this section, there are three tabs: Líneas del pedido (highlighted in blue), Otra Información, and Firma del Cliente. To the right of the client information, a context menu is open over the "Lineas del pedido" tab. This menu lists several options, with the last one, "Relación: sale.order.line", highlighted with a red box. Below the context menu, there is a table titled "Líneas del pedido" with three rows of data. The columns are: Producto, Descripción, Cantidad, and Entregado. The first row shows [FURN_6666] Pantallas de bloque acústico... and [FURN_6666] Acoustic Bloc Screens with 5,000 quantity and 0,000 delivered. The second row shows [FURN_8999] Sofá de tres asientos and [FURN_8999] Three-Seat Sofa Three Seater Sofa with Lounger in Steel Grey Colour with 1,000 quantity and 0,000 delivered. The third row shows [FURN_8888] Lámpara de oficina and [FURN_8888] Office Lamp with 1,000 quantity and 0,000 delivered.

Producto	Descripción	Cantidad	Entregado
[FURN_6666] Pantallas de bloque acústico...	[FURN_6666] Acoustic Bloc Screens	5,000	0,000
[FURN_8999] Sofá de tres asientos	[FURN_8999] Three-Seat Sofa Three Seater Sofa with Lounger in Steel Grey Colour	1,000	0,000
[FURN_8888] Lámpara de oficina	[FURN_8888] Office Lamp	1,000	0,000

Esta relación podemos comprobar que existe en **PgAdmin** ya que si entramos en el **campo 'id'** de la tabla '**'sale_order'**', vemos que existe una **foreign key** hacia la tabla '**'sale_order_line'**:

Foreign key en la tabla padre (one)

Type	Name
Sequence	public.sale_order_id_seq
Function	nextval('sale_order_id_seq'::regclass)
Primary Key	public.sale_order_pkey
Foreign Key	public.procurement_group.procurement_group_sale_id_fkey
Foreign Key	public.purchase_order_line.purchase_order_line_sale_order_id_fkey
Foreign Key	public.sale_order_line.sale_order_line_order_id_fkey
Foreign Key	public.sale_order_option.sale_order_option_order_id_fkey
Foreign Key	public.sale_order_tag.sale_order_tag_rel_order_id_fkey

Si analizamos dicha **foreign key** en la tabla '**'sale_order_line'**', vemos que relaciona el campo '**order_id**' de ésta con el campo **id de la tabla 'sale_order'**:

Foreign key en la tabla hijo (many)

Type	Name
Index	public.sale_order_pkey
Column	public.sale_order_line.order_id
Column	public.sale_order.id

Analizando las relaciones entre las tablas

Cómo saber desde Odoo las relaciones entre las tablas con claves foráneas, y como PgAdmin nos puede ayudar con ello:

Actualizando datos de Odoo desde PgAdmin

Vamos a analizar las relaciones entre las tablas de Odoo y cómo la aplicación PgAdmin nos puede ayudar.

Para ello vamos a irnos a “Ajustes” con el modo de desarrollador activado, para que nos aparezca la sección de “Técnicos”. Aquí podemos ver los “Modelos” que son nuestras tablas:

<input type="checkbox"/> Modelo	Descripción del modelo	Tipo	Modelo transitorio
<input type="checkbox"/> project.update	Actualización del proyecto	Objeto base	<input type="checkbox"/>
<input type="checkbox"/> stock.picking	Albarán	Objeto base	<input type="checkbox"/>
<input type="checkbox"/> account.move	Asiento contable	Objeto base	<input type="checkbox"/>
<input type="checkbox"/> rating.mixin	Calificación Mixin	Objeto base	<input type="checkbox"/>

Si aquí buscamos la tabla “product_template”, si quisiéramos ver una relación *many2one* de *categ_id* “categoría del producto”:

<input type="checkbox"/> Nuevo	Modelos	Producto	
bom_line_ids	Componentes de LdM	one2many	<input type="checkbox"/> <input type="checkbox"/>
can_image_1024_be_zoomed	Puede agrandarse la imagen 1024	booleano	<input type="checkbox"/> <input checked="" type="checkbox"/>
categ_id	Categoría de producto	many2one	<input checked="" type="checkbox"/> <input type="checkbox"/>
color	Índice de Colores	entero	<input type="checkbox"/> <input type="checkbox"/>

Aquí vemos que en la relación, el campo `categ_id` lo está obteniendo de la tabla `product_category`, pero no nos indica la relación en este cuadro, aunque podríamos intuir que será la clave primaria en la tabla `product.category`:

Abrir: Campos

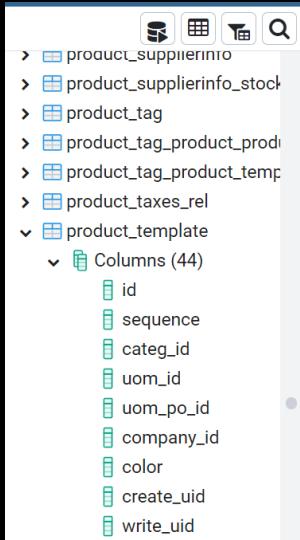
Nombre de campo ?	categ_id	Tipo de campo ?	many2one
Etiqueta de campo ?	Categoría de producto	Campo ayuda ?	
Propiedades Permisos de acceso Varios			
PROPIEDADES BASE			
Requerido ?	<input checked="" type="checkbox"/>	Modelo relacionado ?	product.category
Sólo lectura ?	<input type="checkbox"/>	Al eliminar ?	Restringir
Almacenado ?	<input checked="" type="checkbox"/>	Dominio ?	

Si buscamos esta tabla (`product.category`), en “Técnico” → “Modelos”, vemos que tiene un campo que tiene el ID, e intuimos que la relación puede ser esa:

Nuevo		Modelos
		Categoría de producto
display_name	Nombre mostrado	Carácter <input type="checkbox"/> <input checked="" type="checkbox"/>
filter_for_stock_putaway_rule	stock.putaway.rule	booleano <input type="checkbox"/> <input type="checkbox"/>
id	ID	entero <input type="checkbox"/> <input checked="" type="checkbox"/>
name	Nombre	Carácter <input checked="" type="checkbox"/> <input type="checkbox"/>

Pero una forma más fácil de ver la relación sería haciendo uso de PgAdmin.

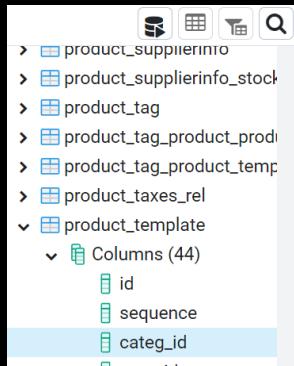
Con PgAdmin, podemos en la tabla “`product_template`”, teniendo elegida la pestaña “Dependents” (Dependientes), vemos todas las relaciones que tiene:



The screenshot shows the pgAdmin interface with the 'Dependencies' tab selected for the 'product_template' table. The table has 44 columns. The 'categ_id' column is highlighted in blue. The dependencies listed are:

Type	Name	Restriction
1..3 Sequence	public.product_category_id_seq	auto
Index	public.product_category_name_index	auto
Index	public.product_category_parent_id_index	auto
Index	public.product_category_parent_path_index	auto
Function	nextval('product_category_id_seq'::regclass)	auto
Foreign Key	public.product_category.product_category_create_uid_fkey	auto
Foreign Key	public.product_category.product_category_parent_id_fkey	auto
Foreign Key	public.product_category.product_category_removal_strategy_i...	auto
Foreign Key	public.product_category.product_category_write_uid_fkey	auto
Primary Key	public.product_category_pkey	auto

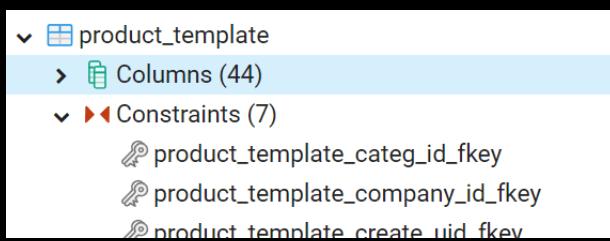
Pero si queremos ver concretamente el campo “categ_id” podemos ver que hay una “Foreign Key” llamada “public.product_template.product_template_categ_id_fkey” siendo product_template la tabla, y “product_template_categ_id_fkey” el campo ID o clave foránea:



The screenshot shows the pgAdmin interface with the 'Dependencies' tab selected for the 'product_template' table. The table has 44 columns. The 'categ_id' column is highlighted in blue. The dependencies listed are:

Type	Name	Restriction
Foreign Key	public.product_template.product_template_categ_id_fkey	auto
Rule	_RETURN ON public.vendor_delay_report	normal
Rule	_RETURN ON public.report_pos_order	normal

Si nos vamos a las “Constraints” (Restricciones) de la tabla observamos la clave foránea que acabamos de ver:



The screenshot shows the pgAdmin interface with the 'Constraints' section for the 'product_template' table. The constraints listed are:

- product_template_categ_id_fkey
- product_template_company_id_fkey
- product_template_create_uid_fkey

Si pulsamos ahora en “Dependences” (Dependencias), nos dice que la relación es de “public.product_template.categ_id” con el “public.product_category.id”, de modo que aquí se ve más fácilmente lo que antes solo intuíamos:

Type	Name
Index	public.product_category_pkey
Column	public.product_template.categ_id
Column	public.product_category.id

Si nos vamos a la tabla “product_category” y abrimos la pestaña “Dependents” (Dependientes), vemos que tiene un campo “id” y con esto vemos que esta es una forma más rápida de analizar cual es la relación de los campos entre tablas, con una Foreign Key:

Type	Name ▾	Restriction
Foreign Key	public.stock_warehouse_orderpoint.stock_war...	normal
Foreign Key	public.stock_valuation_layer.stock_valuation_la...	normal
Foreign Key	public.stock_route_categ.stock_route_categ_ca...	normal
Foreign Key	public.stock_putaway_rule.stock_putaway_rule...	normal
Foreign Key	public.product_template.product_template_cat...	normal
Foreign Key	public.product_pricelist_item.product_pricelist_i...	normal
Primary Key	public.product_category_pkey	auto
Sequence	public.product_category_id_seq	auto

Haciendo backups y restaurando la base de datos

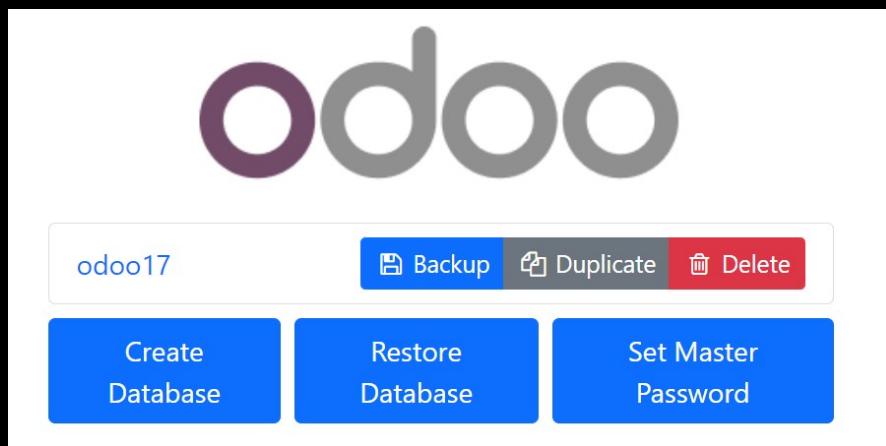
Las copias de seguridad y restauraciones de la base de datos pueden hacerse de **muchas formas** (**incluso** se pueden crear **scripts** para automatizarlas). Aquí explicaremos las **formas más comunes** para nuestro servidor que lo tenemos **en local** (**también** podría hacerse backups si trabajamos en la **nube**):

1. Para **bases de datos pequeñas**:

Con el propio gestor de bases de datos de la interface que nos proporciona Odoo para hacer **backups**. Para ello, simplemente tendremos que entrar en el **navegador** en la siguiente web:

<http://localhost:8069/web/database/manager>

Interface que ofrece Odoo para hacer backups/restore



Esta elección es ideal. Nos dará la opción de **crear un .zip**, con la **carpeta filestore** (donde se guardan los adjuntos) o un **pg_dump sin la carpeta filestore**. La **restauración** podría hacerse desde la **misma ventana**.

2. Para **bases de datos más grandes**:

Lo ideal sería hacer uso del comando `pg_dump` que ofrece PostgreSQL y más concretamente con el uso de un **compresor**.

En su web oficial (<https://www.postgresql.org/docs/9.5/backup-dump.html>) nos indican como hacerlo **con el terminal**:

- Backup: `pg_dump dbname | gzip > filename.gz`

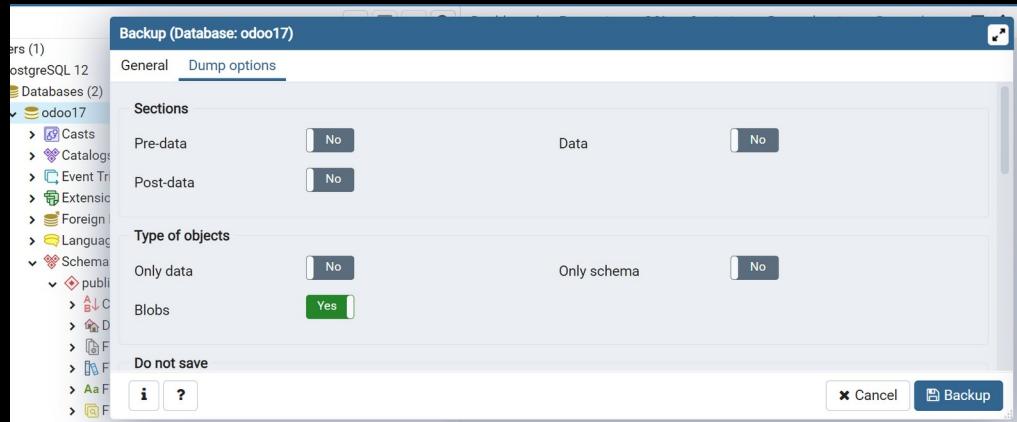
- Restauración: **gunzip -c filename.gz | psql dbname**

3. Otras formas de hacer backups e incluso de automatizar el proceso:

Se pueden hacer copias de seguridad de bases de datos desde:

- La aplicación PgAdmin:

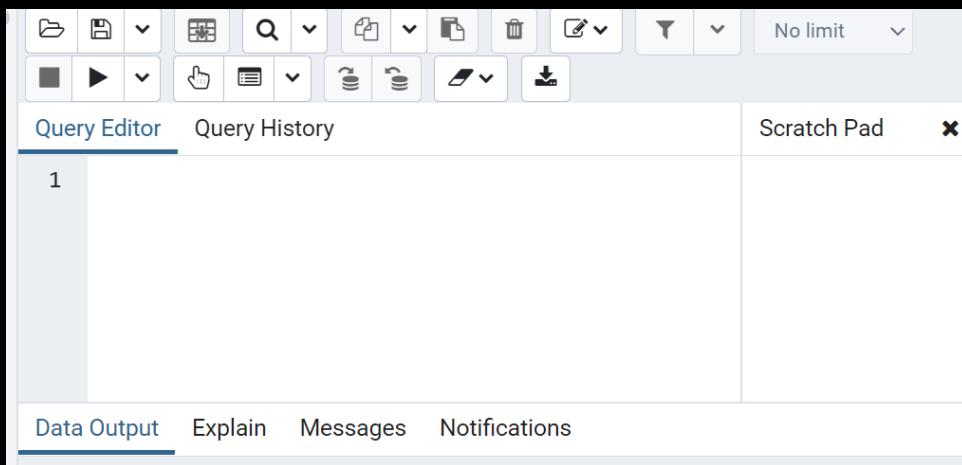
Simplemente tendremos que pulsar con el botón derecho sobre la base de datos a la que queramos crearle una copia de seguridad y después en 'backup'.



- Con **módulos de terceros**, como puede ser '**Database auto-backup**', descargable desde la web oficial de Odoo. Una vez instalado el módulo, cuando entremos en modo desarrollador, aparecerá un **nuevo menú dentro de Técnicos / 'estructura de la base de datos'** que se denomina '**copias de seguridad automatizadas**'. Este nuevo menú nos permitirá configurar **copias de seguridad automáticas** de nuestra base de datos.

Con la herramienta '**Query Tool**' podremos, por ejemplo:

- **Visualizar** un dato en la base de datos: Por ejemplo, realizar una consulta que nos devuelva un listado con las tablas de la base de datos de Odoo ordenadas de mayor a menor tamaño total (excluir las que tengan tamaño 0), seleccionando los datos de pg_catalog.
- **Modificar** un dato de la base de datos.
- **Insertar** un dato en la base de datos.



https://www.pgadmin.org/docs/pgadmin4/latest/query_tool.html

https://www.odoo.com/es_ES/

<https://www.pgadmin.org/>

<https://www.postgresql.org/>

SISTEMA DE GESTIÓN EMPRESARIAL

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Adaptación de Odoo a empresas

Módulo de Odoo Studio

Modificaciones en formularios, informes y demás objetos

Creación de módulos nuevos

Actualización de Odoo

Con la finalidad de beneficiarnos de las **últimas funcionalidades**, correcciones de **seguridad** o **errores**, y mejoras de **rendimiento**, es recomendable actualizar de Odoo cada cierto tiempo.

En Odoo Cloud, las actualizaciones son **automáticas**, por lo que todo lo que trataremos hace referencia a una **instalación propia**.

La base de datos en la actualización

Cuando hablamos de actualizar Odoo, nos referimos a **actualizar el código fuente**, no su base de datos, puesto que actualizar la base de datos es algo más complejo, que implicaría migración de datos, entre otras acciones complejas.

De hecho, actualizar Odoo no causa ningún cambio directo sobre la base de datos de éste, podríamos volver incluso a la versión anterior si la actualización no nos convenciese y esto no afectaría a los datos de la base de datos. Realmente lo que se hace al actualizar es **reinstalar la nueva versión sobre la actual**, dejando **intacta la base de datos**.

Pasos para actualizar la versión instalada de Odoo:

1. Descargar la versión a instalar desde la página oficial de Odoo

https://www.odoo.com/es_ES/page/download.

2. Hacer un *backup* o copia de seguridad de la **base de datos, es altamente recomendable**. Aunque hayamos comentado que la actualización no afecta a esta, **es preferible** siempre tener una copia de seguridad almacenada en lugar seguro.

3. Instalar la nueva versión, que se puede hacer de varias formas:
 - Con un paquete instalador: si originalmente se usó este método de instalación, entonces se podrá actualizar igual. Es el método más fácil, simplemente hay que descargar el instalador (paso 1) y **ejecutarlo**. Esto **se instalará sobre la actual**. Ya solo queda reiniciar el servidor y Odoo.
 - Con archivos de código fuente (**Github**): si por el contrario, se usó un **clon de Github** para instalar Odoo, la actualización se hará **a través de git**, con los comandos git:

git fetch

git rebase --autostash

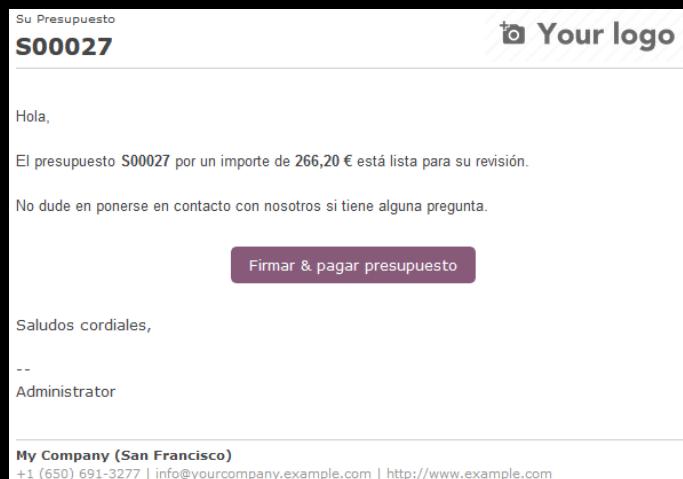
El último comando (`git rebase -autostash`) puede **generar conflictos** con el código fuente si hemos **editado el código fuente** de Odoo. Habrá que resolverlos manualmente y **decidir qué código se queda**. Finalmente, habrá que **reiniciar el servidor y Odoo**.

- Con Docker, se puede ejecutar directamente una nueva versión, solo que hay que copiar los **archivos** de la ruta '`(var/lib/odoo/filestore/)`' en la **nueva versión**.

Adaptando Odoo a la empresa

Veremos como **sin saber programar en Python** es posible **crear o modificar un módulo** en Odoo.

Algunas de las formas más básicas de adaptar Odoo a nuestra empresa es modificando las plantillas de los documentos (facturas, presupuestos, etc.) y las plantillas de correos que se envían. Por ejemplo, cuando enviamos un presupuesto por e-mail, por defecto se envía la siguiente información con el siguiente presupuesto adjunto:



Si quisiéramos **agregar un logo** a nuestra empresa, deberíamos hacerlo desde:

ajustes / opciones generales / compañía / actualizar información

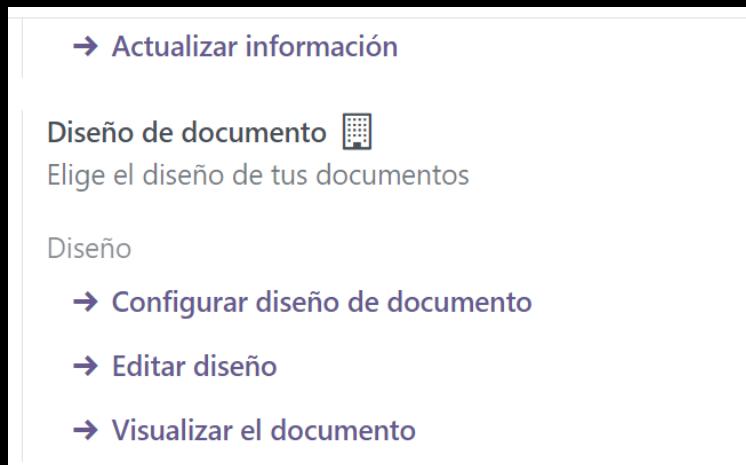
Modificando la información de la compañía

A screenshot of the Odoo company information update screen. The top navigation bar has tabs for "Opciones generales", "Usuarios y compañías", "Traducciones", and a plus sign. The "Ajustes" tab is selected. On the left, there is a sidebar with "Generales" and "/ Contabilidad". The main area shows company details: "My Company (San Francisco)", address "Gran vía 14, 28200 - Madrid, Madrid (ES), España", and NIF "US12345671". Below the details is a button "→ Actualizar información".

Si quisieramos **modificar la plantilla del documento** pdf (factura, prefactura, etc.), deberíamos hacerlo desde:

ajustes / opciones generales / Documentos de negocio

(En versión 17 es: ajustes / opciones generales / **compañías / configurar diseño del documento**)



Y si quisieramos **modificar el texto de la plantilla** que se usa para enviar los **e-mails**, deberíamos hacerlo desde los ajustes, en el menú técnico/correo electrónico/plantillas:

Modificando los e-mails

Plantillas de correo electrónico			
Nuevo	Aplica a	Usua...	Descripción de la plantilla
<input type="checkbox"/> Compra: Recordatorio de proveedor	Pedido de compra		Enviado a los proveedores antes de la llegada prevista, según la configuración de la orden de compra
<input type="checkbox"/> Compra: Solicitud de cotización	Pedido de compra		Enviado manualmente al proveedor para solicitar una cotización
<input type="checkbox"/> Compra: orden de compra	Pedido de compra		Enviado al proveedor con la orden de compra como archivo adjunto

Pequeñas adaptaciones en Odoo

Veremos cómo realizar algunas adaptaciones básicas de Odoo para nuestra empresa y su resultado, cómo editar la plantilla, tanto del formulario como del correo que se envían por defecto en Odoo, en este caso, de un pedido de venta.

Para empezar, el ejemplo que seguiremos será la factura en un presupuesto que enviaremos a un cliente. Comentar que cuando entramos en venta nos permite:

- informar el logo de la compañía y más datos,
- diseñar el informe.

Si por cualquier cosa lo cerramos y no lo encontramos, lo trataremos desde su estándar. Hasta, en este ejemplo, el documento que se envía al usuario de presupuesto es el que hay por defecto, con el PDF adjunto. No tiene ni logo.

Le vamos a hacer un cambio básico. Para cambiar el logo de nuestra empresa:

1. Vamos a **ajustes**.
2. En **opciones generales** -> actualizar información de la compañía.
agregaremos el logo de nuestra compañía. Y pulsaremos en guardar.

Volveremos a **ajustes** para ahora **modificar la plantilla**. Lo podemos hacer desde la pantalla anterior, pero en la sección documentos de negocio (versión 16), vamos a **configurar diseño de documento**. Y ya si os fijáis nos ha puesto unos **colores acorde con el logo**. **Podremos cambiarle** algunas opciones muy básicas también:

- Los colores, pues en lugar de negro, pueden ser en azul.
- El tipo de letra, tenemos varias opciones.
- El lema de la compañía,
- El pie de página, que en este caso no lo pondremos.

Y guardamos.

Para la **opción del e-mail** tendremos que entrar a:

técnico → correo electrónico → plantillas

Seleccionaremos la plantilla, que en nuestro caso es del pedido de venta, un e-mail. Y, por ejemplo, al editar, podemos cambiar el contenido. Y guardamos. Ahora, al crear un presupuesto de venta, vamos a ver el **resultado**:

Hemos creado el presupuesto y hemos pulsado el botón de enviar presupuesto por correo. Y en la plantilla del e-mail nos aparece la palabra test. Y en el documento adjunto nos aparecerá la plantilla actualizada, tanto con el logo como los colores y demás cambios que hemos tocado en esa pantalla.

Modificación de los módulos del sistema con Odoo

Odoo Studio es un módulo de Odoo, **creado por** la misma empresa **Odoo**, que nos permite **modificar o crear formularios e informes, personalizar** las pantallas de una manera avanzada con **pivot tables, kanban, Graph**, etc., y muchas más opciones, **sin** usar ni una **Línea de código**; todo mediante una **interfaz gráfica** y un **constructor visual** de arrastrar y soltar. Incluso puede crear **aplicaciones** con una interfaz amigable para **móviles**.

El único inconveniente a destacar es que esta potente herramienta, creada por la misma Odoo S.A., solo es accesible desde la **versión Enterprise, no es Open Source**. Es por ello que para este tema, haremos uso de la **versión de prueba** que Odoo nos ofrece.

Para **instalar este módulo**, bastará con **entrar en la web de Odoo** en nuestra cuenta (crearemos una nueva cuenta si no tenemos) y, a continuación, en ‘**tarifas**’, seleccionaremos Odoo y pulsaremos ‘**Empezar ahora**’. Al ser la **única aplicación, será gratuita** mientras no usemos otra.

Tal y como sabemos, Odoo usa **lenguaje XML para pintar las vistas**. Odoo Studio actualiza/crea esas vistas conforme las vayamos actualizando al arrastrar y soltar las etiquetas con el ratón.

Al **instalar el módulo**, aparece un nuevo **ícono**, que estará **disponible en todas las pantallas**:

Ícono que se habilita al instalar Odoo Studio

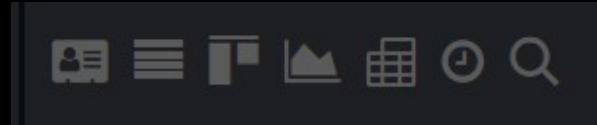


Al pulsarlo en la ventana de ‘Aplicaciones’, nos permitirá, entre otras cosas, **crear una nueva aplicación**.

Al pulsarlo dentro de un módulo, se abrirá el editor de Odoo Studio (ventana en negro) con el que podremos **modificar dicho módulo**.

El menú **vistas tiene 7 vistas**:

1. **Formulario,**
2. **Lista,**
3. **kanban,**
4. **Gráfico,**
5. **Pivote,**
6. **Actividad,**
7. **Búsqueda.**





- para modificar un formulario, pulsaremos sobre el **menú ‘Vistas’**,
- para crear automatizaciones en **‘Automatizaciones’**.

También se pueden implementar **filtros** para **mostrar u ocultar** campos bajo ciertas **condiciones**, entre otras opciones.

4.1. Personalizando un informe con Odoo Studio

En este punto, vamos a **modificar un informe**, denominado **‘imprimir identificación’**, que existe en el módulo de Empleados.

Queremos que, además, aparezcan los siguientes datos:

- el **nombre en rojo**,
- el **sin logo** de la empresa,
- el **móvil** del trabajo,
- el **correo** del trabajo,
- un texto ‘**Hijos:**’ seguido del **número de hijos**, pero solo **si tiene**.

Para ello, al igual que antes, una vez que entramos **en el módulo de Empleados**, abriremos **Odoo Studio**, pero ahora pulsaremos sobre **‘informes’**. **Seleccionaremos el informe** que queremos modificar, que en este caso es **‘imprimir identificación’**, y se nos **abrirá el editor** de Odoo Studio. Ahora, solo nos queda **agregar los campos**.

(Nota: en la versión 17 es distinta la interfaz de usuario; este tema está basado en una versión anterior a la 17; se comentará en algunos casos las referencias a la versión 17).

Finalmente, a los **campos de hijos** (texto 'Hijos:' y valor del número de hijos), hay que agregarle en las opciones la **condición de que solo sean visibles** cuando **tenga hijos**.

Resultado de la modificación del informe

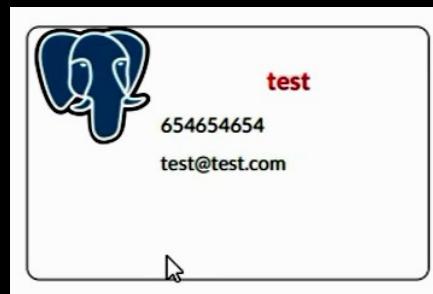
Vamos a comprobar que los cambios que hemos descrito en la documentación funcionan correctamente.

Lo que hemos hecho ha sido modificar el formulario del informe de los Empleados.

Así, por ejemplo, si entramos en este **usuario** que sí tiene tres hijos, al imprimir la identificación que hemos modificado, si recordamos hemos puesto el nombre en rojo, el teléfono, la dirección de correo y el **número de hijos si este tuviese**.



Si, por ejemplo, ahora vamos a otro empleado que no tiene hijos, si imprimimos aparecerá el teléfono, la dirección de correo, y el nombre de los Empleados, pero **no aparecerán los hijos**. Y el nombre de los Empleados se va a poner en rojo.



Con esto comprobamos que los cambios han funcionado correctamente.

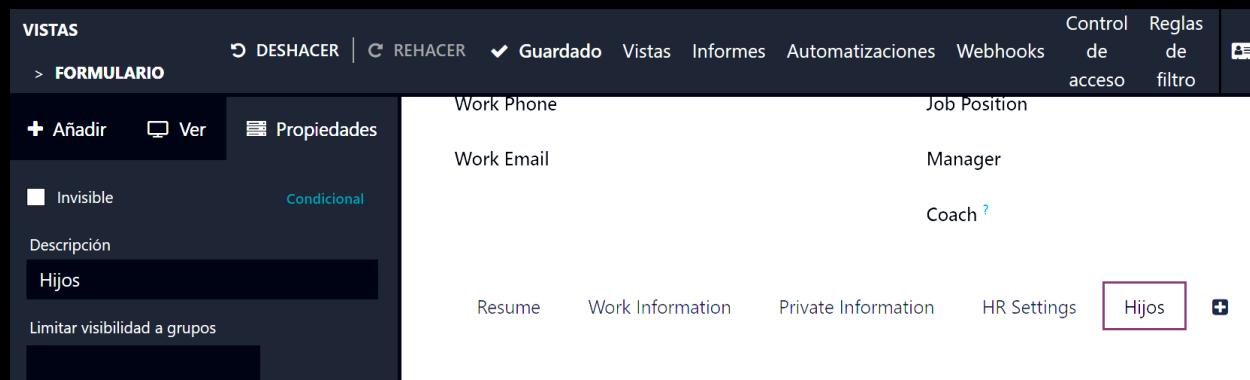
Ejemplo de Personalización de un formulario existente con Odoo Studio

Se requiere modificar la vista ‘**formulario**’ del módulo de Empleados (se pueden editar también la vista kanban, lista o actividad). Concretamente, se necesita agregar una **nueva pestaña** para, por ejemplo, **saber si el empleado tiene hijos**.

Esta nueva pestaña tendrá la pregunta ‘¿Tiene hijos?’ y un botón donde, si está activado, aparecerá la pregunta ‘¿Cuántos?’ y un desplegable con **valores entre 1 y 5 y la opción ‘más’**, que será obligatorio informar.

Para ello, una vez que entremos en el **módulo de Empleados**, abriremos Odoo Studio y seguidamente pulsaremos sobre “vistas” y “formularios” o sobre el icono de la vista de formulario.

A continuación, pulsaremos sobre el siguiente “+”, de la lista horizontal de pestañas, para agregar la **nueva pestaña**, dándole el **nombre “Hijos”** a esa nueva pestaña.



Después, agregaremos un **nuevo campo** de tipo ‘casilla de verificación’ con las siguientes propiedades, que se eligen en la pestaña propiedades del elemento del formulario:

- En descripción tendrá el texto “¿Tiene hijos?”.

Además, añadiremos el desplegable (selección) ‘¿Cuántos?’ con los **valores entre 1 y 5**, y la **opción ‘más’** con las siguientes propiedades:

- En descripción tendrá el texto “¿Cuántos?”.
- En “Invisible” tendrá marcada la casilla, y tendrá una condición, que el elemento “¿Tiene hijos?” no esté marcado (no es establecido), de modo que no se muestre si no tiene hijos:

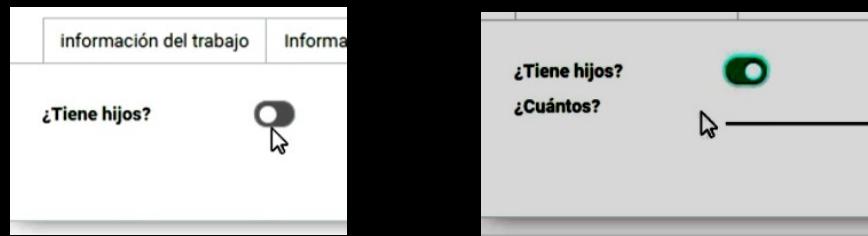
Propiedad y condición para ‘invisible’ del campo ‘¿Cuántos?’

The screenshot shows a form configuration interface. On the left, there's a sidebar with options like 'Añadir', 'Ver', 'Propiedades', 'Invisible' (unchecked), 'Requerido' (checked), 'Sólo lectura', 'Descripción', '¿Cuántos?', 'Cuadro de información de Ayuda', and 'Widget'. The 'Propiedades' tab is selected. In the main area, there's a section for '¿Tiene hijos?' with a checkbox. Below it, there's a '¿Cuántos?' input field. A modal window titled 'Edit Condition' is open, containing the text 'Combinar todos de las siguientes reglas:' followed by a condition entry: '¿Tiene hijos? no es establecido'. There's also a link 'Nueva regla'.

Resultado de la modificación

Vamos a comprobar que las acciones que hemos descrito en la teoría funcionan correctamente:

Para ello, nos vamos a ir al módulo de Empleados, vamos a crear un **nuevo empleado** al que llamaremos **Miguel**. Y vemos ya que **tiene la pestaña hijos**. Si la pulsamos vemos que solo aparece la pregunta de **¿tiene hijos?** y el **activador switch** que si lo pulsamos **hace que aparezca la pregunta de ¿cuántos?**



Si **intentamos guardar** el empleado sin informar esta pregunta de ¿cuántos? nos mostrará un **mensaje de que el siguiente campo** tiene que **ser informado**, porque dijimos que si este estaba marcado debería ser **obligatorio**:

The screenshot shows a software application window with a navigation bar at the top containing tabs: 'información del trabajo', 'Información Privada', 'Configuración RRHH', and 'Hijos'. The 'Hijos' tab is currently selected. Below the tabs, there are two questions: '¿Tiene hijos?' with a checked radio button and '¿Cuántos?' with an empty input field. To the right of the input field for '¿Cuántos?' is a small red asterisk indicating it is a required field. A modal dialog box is displayed, titled 'Los siguientes campos son inválidos:' (The following fields are invalid:), with a single bullet point: '• ¿Cuántos?'. The background of the dialog box is pink.

Entonces lo desplegamos, **informamos que tiene tres hijos**, por ejemplo, y ahora sí nos permitirá **guardar** el empleado Miguel.

Creación de módulos propios: Creación y diseño

Odoo Studio, además de permitirnos modificar módulos existentes, también nos otorga las herramientas necesarias para la creación de **nuevos módulos sin tener que programar**. En este punto, vamos a simular que tenemos una **empresa de alquiler de oficinas** y necesitamos una aplicación a medida que nos facilite algunas tareas.

Es de sobra sabido que existen infinidad de tipos de empresas, cada una con sus peculiaridades y necesidades específicas, por tanto, está en la **capacidad de adaptación del ERP**, una de sus principales ventajas competitivas.

Esto, **combinado** con la labor de **consultoría**, formarán el tandem perfecto para adaptar los sistemas a las **necesidades concretas** de las empresas.

Creación del módulo

El módulo se llamará ‘Gestión de alquileres’. Para crearlo, desde la **pantalla de las aplicaciones**, pulsaremos sobre el **ícono de Odoo Studio** y nos aparecerá la opción de **crear un módulo nuevo**.

Icono para crear un nuevo módulo



Al pulsarlo, Odoo nos guiará en la creación. En primer lugar, nos requerirá **un nombre y un ícono**:

Editando el nombre y el ícono del nuevo módulo

A screenshot of the Odoo Studio interface. On the left, a light gray panel titled "Crea tu aplicación" contains a text input field with placeholder text: "Elige un nombre de aplicación por ejemplo: Inmobiliaria". At the bottom are two small navigation arrows. On the right, a dark blue panel titled "Diseña tu ícono" features a yellow diamond-shaped icon on a blue background. To its right are three color swatches (blue, yellow, and black) each with a dropdown arrow, and below them is a button labeled "o subirlo".

A continuación, nos pedirá el nombre de nuestro **primer objeto (primer menú)**, que será '**Oficinas en alquiler**'. Con esto ya se habría **creado el módulo** y, de **forma automática**, también las **tablas custom** en la **base de datos** (sus nombres **empiezan por x_**).

Diseñando el formulario: El formulario de las oficinas en alquiler incluirá:

- El nombre: Viene **por defecto** en el formulario.
- La foto: Hay que arrastrar un nuevo **campo 'imagen'**.
- Precio del alquiler: Arrastrar un nuevo campo '**monetario**'.
- El número de puestos de trabajo: Hay que arrastrar un nuevo campo '**número entero**'.
- La dirección: Hay que arrastrar un nuevo campo '**texto**'.

Diseñando el formulario

The screenshot shows a software interface for designing forms. At the top, there's a navigation bar with 'VISTAS > FORMULARIO', 'DESHACER', 'REHACER', 'Guardado' (with a checkmark), and 'Vistas'. Below the navigation is a toolbar with buttons for '+ Añadir' (Add), 'Ver' (View), and 'Propiedades' (Properties). A sidebar on the left is titled 'Nuevos campos' (New fields) and lists several field types with icons: Texto (Text), Entero (Integer), Decimal, HTML, Monetario (Monetary), Fecha (Date), Fecha y Hora (Date and Time), Casilla de ver... (Checklist), and Selección (Selection). To the right, the main area displays a form with three fields: 'Nombre' (Name) with a value of 'Oficina de Alquiler', 'Precio' (Price) with a value of '0,00', and 'Número de puestos de trabajo' (Number of workstations) with a value of '0'. There is also a placeholder 'Dirección' (Address).

Creación de módulos propios: Gestión y vistas

Gestionando el estado del alquiler en la barra de estado

Además de lo implementado anteriormente, vamos a hacer que en la barra de estado aparezca el **estado del alquiler**, que puede ser:

- disponible,
- alquilado,
- a renovar.

Para ello, en el mismo formulario, si nos fijamos en la parte superior aparece el texto:

add a pipeline status bar

Al pulsarlo, nos aparece el siguiente pop-up:

Configurando los estados de la barra de estado



Bastará con **informar de los estados** con los valores que queremos y, finalmente, **confirmar**.

Barra de estado



Creando las oficinas y mejorando la vista de lista

Ya estaremos en disposición de crear las oficinas en alquiler. Para ello, entraremos en nuestro módulo '**Gestión de alquileres**' y pulsaremos sobre el botón de '**Crear**'.

Informando el formulario

Precio del alquiler	3000	Foto
Número de puestos	50	
Dirección	C/ Real, 1	

La vista de lista queda pobre con el 'nombre' como único dato.

Podremos agregarle **más datos** con Odoo Studio entrando en la **edición de la vista 'lista'** (en lugar del formulario) y **arrastrando los campos existentes**.

Ejemplo agregando una vista kanban

Para crear en nuestro nuevo módulo una vista kanban para visualizar más rápidamente las **oficinas que están alquiladas y las que no**, bastará con entrar en nuestro **módulo**, pulsar el botón de Odoo Studio, seleccionar '**Vista**' y, finalmente, la **vista kanban**.

A continuación, en el menú '**Ver**', agruparemos las propiedades **por su estado** (Pipeline status bar).

De esta forma, dispondremos de una vista kanban en la que, rápidamente, podremos distinguir las **oficinas que están alquiladas y las que no**:

Vista Kanban de las oficinas

The screenshot shows the Odoo Kanban view for 'Gestión de alquileres' (Rental Management). The top navigation bar includes a menu icon, the module name 'Gestión de alquileres', and a link 'Oficinas en alquiler'. Below the header, there are two buttons: 'CREAR' (Create) and 'IMPORTAR' (Import). The main area displays two columns: 'Disponible' (Available) and 'Alquilado' (Rented). The 'Disponible' column contains one record: 'Oficina Málaga' with a rating of 2 stars. The 'Alquilado' column contains one record: 'Oficina Madrid' with a rating of 1 star.

Disponible	Alquilado
Oficina Málaga ☆☆	Oficina Madrid ★☆

Crear informes estadísticos

Odoo Studio también nos permite crear informes estadísticos con **vistas gráficas**.

Por **ejemplo**, si quisiéramos tener un gráfico para analizar de un solo vistazo el **número de puestos de trabajo** en cada **oficina**, podríamos hacerlo. Para ello deberíamos (igual que con la vista kanban):

1. entrar en nuestro **módulo**,
2. pulsar el botón de **Odoo Studio**,
3. seleccionar 'Vista', pero esta vez entraremos en '**Graph**'.

Una vez activada, podremos salir del editor, puesto que ya se nos habrá habilitado el icono de vista gráfica, el cual, al pulsarlo, nos dará varias **opciones de visualización** a la que podremos aplicarle **filtros**.

Diseñar una plantilla de informe

Esta herramienta también nos ayuda en la creación de informes. Ahora, si quisiésemos crear un informe con los **datos** de las oficinas, podríamos hacerlo en la **sección 'informes'**. Al entrar por primera vez en nuestro módulo, no tendremos ningún informe creado, por lo que pulsaremos sobre el botón 'Crear' y crearemos uno 'en blanco' o vacío.

Ahora **arrastraremos los siguientes campos**:

- foto,
- nombre,
- dirección,
- precio del alquiler,
- número de puestos.

Resultado de la modificación plantillar

Vamos a comprobar el funcionamiento del módulo que hemos creado en este tema. Este es el módulo que vemos con el icono y el nombre de gestión de alquileres.



Al pulsar vemos esta vista de lista a la que le agregamos, a partir de solo con el nombre, el precio de alquiler y el número de puestos.

Oficinas en alquiler		Buscar...	Q						
		Filtros	Agrupar por	Favoritos	1-2 / 2	<	>	grid	list
<input type="checkbox"/>	Name		Precio del alquiler	Número de puestos					
<input type="checkbox"/>	Oficina Málaga		3.000	50					
<input type="checkbox"/>	Oficina Madrid		5.000	40					

Si entramos por ejemplo a la oficina de Málaga, vemos el formulario con el nombre, precio de alquiler, número de puestos, la dirección y la foto.

También podemos ver que agregamos **estados**, si está disponible, alquilado o a renovar.

El informe de impresión que hemos creado, estará disponible para el informe de oficina de alquiler.

Y podemos ver que al imprimirla se nos crea un **PDF** con la imagen y algunos datos que le hemos indicado, como son la imagen, el nombre, la dirección, el precio y el número de puestos.

También hemos visto las **visualizaciones en kanban**, está, que **agrupamos** por disponible o por alquilado o agrupamos por estado.

Y también vemos una **vista gráfica**; podemos ver por ejemplo la oficina de Madrid de Málaga en un gráfico por el número de puestos.

Hemos comprobado todas las funcionalidades que hemos creado con este nuevo módulo.

Apuntes

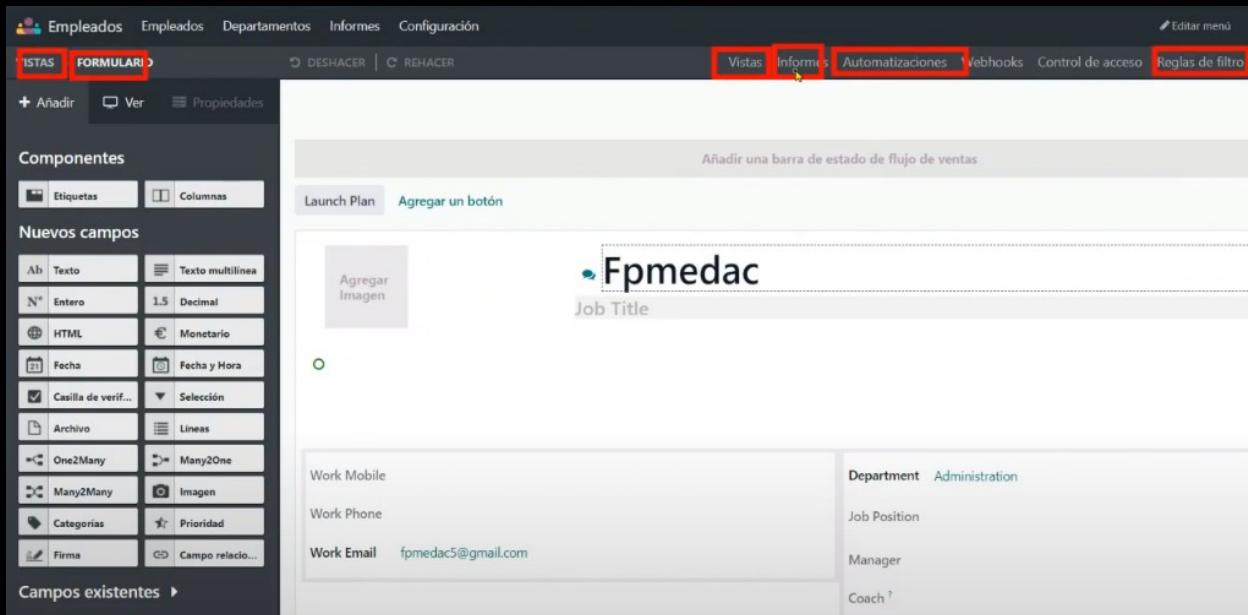
Versión 17 de Odoo

Para aprobar Odoo Studio vamos a irnos al módulo Empleados.

Dentro de Empleados, puedo darle a la lista de Odoo Studio.

En la lista de formulario dentro de Empleados, ¿qué voy a poder hacer? Voy a poder ir a:

- distintas listas,
- a los informes,
- a la regla de filtro,
- a automatizaciones,
- etc.

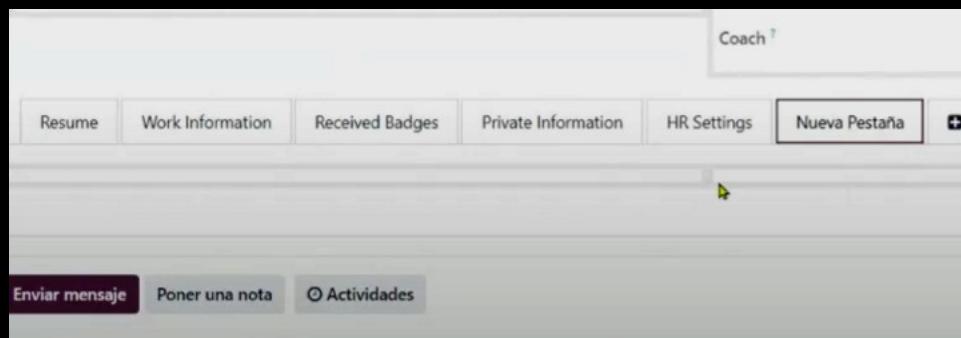


Ahora vamos a estar alternando entre listas e informes. ¿A qué corresponde todo esto? Corresponde a lo que a mí me aparece cuando yo entro dentro de empleado, en principio me aparece:

- el nombre,
- la imagen,
- distintos campos,
- distintas pestañas,

Nosotros podemos añadir otras pestañas. Vamos a crear una **nueva pestaña con campos**. ¿Con qué? Utilizando Odoo Studio.

¿Para qué? Para personalizar; En concreto, algo que el cliente me puede llegar a pedir o necesite. Le voy a dar una nueva pestaña.



En la nueva página creada hay una nueva pestaña.

Entonces, ¿qué voy a hacer en la nueva pestaña? Este es el espacio que tengo podemos añadir los campos que queramos. Entonces, si le doy a “**añadir**” dentro de la lista de formulario, me van a aparecer todos los posibles campos que yo pueda incorporar o los campos existentes.

Vamos a poner una casilla de verificación. Y voy a ponerle “**¿Tiene departamentos?**” Le voy a poder decir que es un tipo booleano, con una casilla de verificación. Le voy a decir que es un intercambiador.

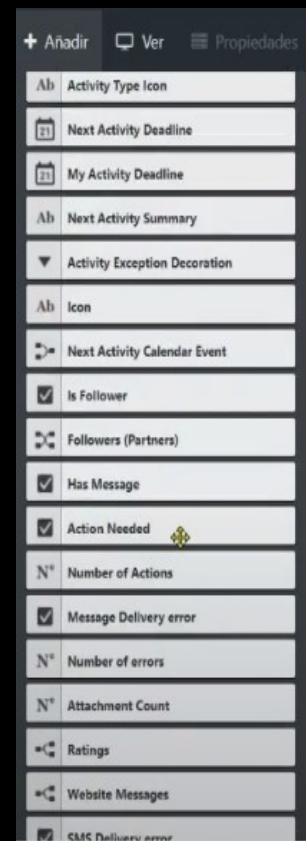
Puedo meter otro campo, por ejemplo, el campo **selección**. Voy a seleccionar, uno, dos, o tres **departamentos** o “más departamentos”.

Voy a ponerle además “**¿Tiene hijos?**” Pues lo puedo clickear.

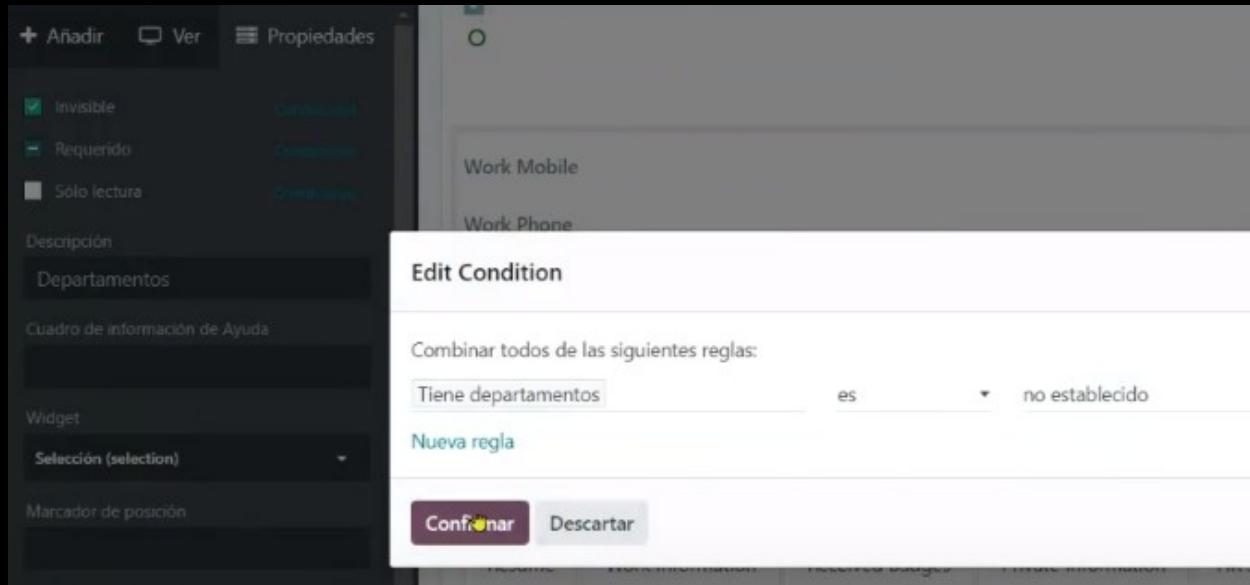
Esto lo pongo en una nueva pestaña. La nueva pestaña debería aparecer. Si yo cierro Odoo Estudio, me vengo al formulario empleado, me encuentro las distintas pestañas y me encuentro la nueva pestaña. Y en esta nueva pestaña me aparece si tiene departamentos, Y me aparece la selección que yo he puesto.

Puedo llegar a **personalizarlo más**, de modo que puedo poner que este departamento sea un campo que **no es visible**, hasta que no active que tiene más departamentos. Para ello voy a Odoo Estudio y le digo que en mi nueva pestaña, los departamentos, pues que este departamento sea un **campo invisible**. ¿Qué ocurre? Que ahora no me aparece. Tengo que darle en **ver** que muestre los elementos invisibles.

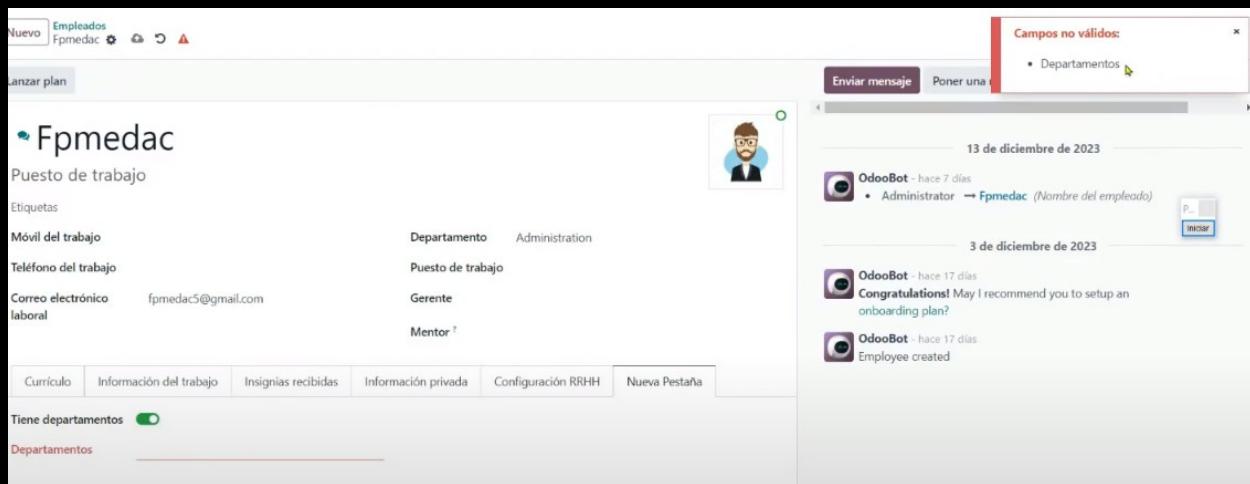
En **propiedades** sigo trabajando con este elemento. Esto es invisible y además le voy a decir que es **requerido**. Bien, pero el que sea invisible, es en función de una regla. Tengo que poner:



- En invisible, la condición de que departamentos es no establecido.
- Y en requerido, le pongo un condicional de que cuando tiene departamentos... No esté establecido. Confirmo. Y si no, estoy poniendo las condiciones.

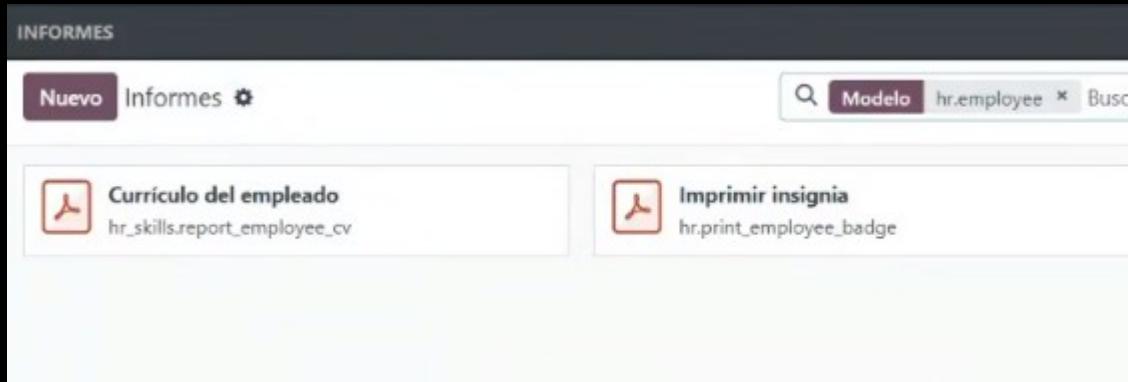


Voy a probarlo: Le voy a dar a cerrar. Me voy a una pestaña: Tiene departamentos, está activo. Me aparece el nuevo campo y me dice que es **no válido porque está esperando un valor**. Y ahora me permitiría guardar.



Realmente tiene una potencia interesante el uso de Odoo Estudio para estar haciendo vinculaciones con campos, incluyendo nuevos campos.

De vista, en el formulario vamos a irnos a **informes**:

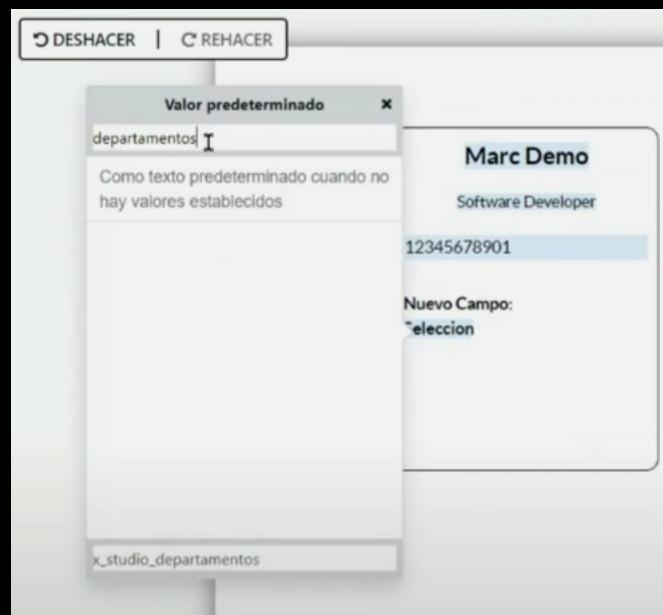


Dentro de informes, los **informes que están incluidos en Empleados**, son uno para el currículum y otro para imprimir insignia. Estos informes los puedo **personalizar**. De modo que me voy al informe en cuestión y **no me aparecen los campos** como viene en los apuntes, pero con la **nueva versión 17** es tan fácil como darle al **slash**, y me aparece la parte como de backend.



Entonces, las estructuras habituales que me puedo encontrar, pues aparecen, para yo ir **configurando todo el informe**, como quiera:

Puedo ir a buscar un campo. ¿Qué campo? Pues tiene "Departamentos" creado anteriormente. Entonces, ¿qué texto aparece? Pues departamento por defecto, puedo poner lo que quiera que aparezca en el informe en el caso de que no tenga nada.



Puedo incluir el campo nuevo. Lo voy a poner en rojo. Y lo guardo en el informe. He personalizado el informe.

Voy a intentar **imprimir los distintos datos** que tengo. Lo señalo todo. Le digo que imprima. Va a utilizar la plantilla PDF que yo tenía. Esa plantilla PDF la he modificado. Y aparece ese nuevo campo. Entonces, si tiene valor, pues el valor, que no, pues el valor que le haya puesto por defecto. Y me da la potencia de poder generar unos reportes, unos informes adecuados a las necesidades que vaya pidiendo el departamento o el siguiente puesto.



Pues con esto estoy personalizando el formulario y personalizando el informe.

Crear módulos propios

Dentro de la **pantalla principal** activo Odoo Studio. Al activar Odoo Studio **me permite crear una nueva** aplicación.

Le doy a siguiente. Podemos poner de **nombre**: Alquiler casas nuevas.

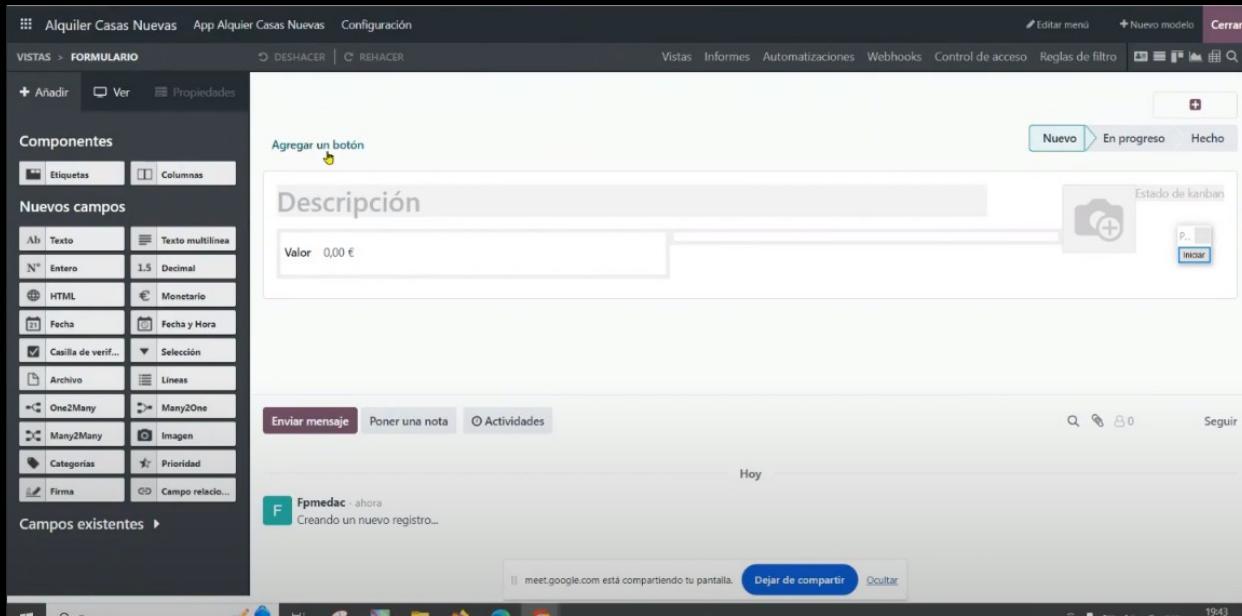
Se puede subir, si quiero, una **imagen**. Tener en cuenta que todo lo que ponga va a tener cierta repercusión a la hora de hacer los informes. Le doy a siguiente. App nueva. Puedo hacerlo con un modelo nuevo o con un modelo existente.

Si yo lo que voy a hacer es desarrollar vistas, informes, todo eso que estaba trasteando sobre un modelo en concreto, donde mi cliente quiere más detalles, lo puedo elegir. Modelo existente y elijo el modelo en cuestión.

Si yo lo que estoy haciendo es una aplicación de cero con todos esos datos nuevos, Pues, le digo que le quiero un modelo para esta aplicación en cuestión. Entonces, le doy a siguiente. Y, todo lo que pueda estar configurando de principio, pues, me ahorra el trabajo de estar incluyendo después, Porque voy a decir que tengo un valor monetario, que permita etapas, si quiero un calendario, que tenga imágenes, en fin, pues, todo esto lo puedo ir incluyendo.

<input type="checkbox"/> Detalles de contacto Obtener los campos de contacto, teléfono y correo electrónico en los registros	<input type="checkbox"/> Asignación de usuario Asignar un responsable a cada registro
<input type="checkbox"/> Fecha y calendario Asignar fechas y visualizar registros en un calendario	<input type="checkbox"/> Rango de fechas y Gantt Definir las fechas de inicio y fin y visualizar los registros en un gráfico de Gantt
<input checked="" type="checkbox"/> Etapas del pipeline Organice y visualice registros en una canalización personalizada	<input type="checkbox"/> Categorías Categorizar los registros con etiquetas personalizadas
<input checked="" type="checkbox"/> Foto Agregue una imagen a un registro	<input type="checkbox"/> Líneas Añada detalles en los registros con una vista de lista insertada
<input type="checkbox"/> Notas Escribe notas o comentarios adicionales	<input checked="" type="checkbox"/> Valor monetario Establecer un precio o costo en los registros
<input checked="" type="checkbox"/> Orden Personalizado Ordenar manualmente los registros en la vista de lista	<input checked="" type="checkbox"/> Chatter Envíe mensajes, registre notas y programe actividades
<input checked="" type="checkbox"/> Archivando Archivar los registros obsoletos	

De modo que le doy a crear aplicación y me aparecería. ¿Dónde estoy? Pues, en mi aplicación Alquiler casas nuevas. Dentro de la **vista de formulario**. Yo puedo cambiar a vista de informe o puedo cambiar con las reglas, a poner lo que quiera:



¿Qué voy a hacer en esta vista de formulario? Voy a poner:

- el **nombre**.
- Un valor, el **precio**. Este precio, porque lo he puesto al principio.
- Y si no, hay **campos** que me faltan, pues, lo voy a ir arrastrando.
- ¿Qué más quiero? Pues, puedo ponerle un **texto** donde ponga dirección. Donde ponga dirección.
- Un **texto** donde ponga el número de puestos o de gente que permite.
- Le añado otro que sea un entero, **plazas disponibles**.

Estoy componiendo todo lo que es mi vista de formulario.

Si yo cierro, esto se ha guardado. Si me voy a alquiler casas nuevas, y entro. Le voy a dar a nuevo. Y puedo crear. Casa uno. Y ponga una foto. Pongo el precio que quiera. Le doy a guardar. Y así con las casa que necesite.

The screenshot shows a kanban-style interface for managing new house listings. At the top, there are buttons for 'Nuevo' (New), 'App Alquier Casas Nuevas', and 'Casa 1'. A gear icon indicates settings. Below this, a navigation bar has 'Nuevo' (highlighted in green), 'En progreso' (In progress), and 'Hecho' (Done). The main area displays a card for 'Casa 1' with the following details:

Dirección	Gran Vía 33
Precio	€1.200,00
Plazas disponibles	4

To the right of the card is a small thumbnail image of a modern white house.

Pues si yo me voy y tengo la vista de kanban, que son como con fichitas.

Puedo darle a modo lista. Me aparecen mis distintas casas con los campos que yo quiera. Esto es personalizable.

La vista kanban es personalizable. De hecho, yo puedo editar. Le voy a decir que sea **alquilada** o que esta sea **vacía**. Y son los **estados** que yo voy a poder mantener. Y si quiero crear nuevas etapas, lo puedo poner en **vendida**. Y lo guardo. Entonces, si yo entro en una casa, puede ser alquilada o vendida. Si la guardo, habrá cambiado.

The screenshot shows a kanban view of three house listings. The top row shows summary counts for each status: 'Nuevo' (2.700), 'Alquilada' (0), 'Vacía' (0), and 'Vendida' (0). Below this, the cards are listed:

Casa	Estado	Precio
Casa 3	Alquilada	600,00 €
Casa 2	Vacía	900,00 €
Casa 1	Nuevo	1.200,00 €

Each card includes a small thumbnail image of the house and a circular edit button.

Puedo hacer cambiar el estado también en el modo kanban. Si lo hago en el modo lista, los edito.

Formulario:

Puedo irme al modo lista. Puedo cambiar de sitio los distintos elementos. Puedo coger e incluir nuevos campos que se vayan incorporando.

Modo kanban: Y puedo ir añadiendo, personalizando un poco más toda esta parte de los elementos que aparecen. Si es favorito o no favorito. Y los campos que yo quiera incluir.

Modo pivot: puedo incluir los **datos de cada uno de los objetos** que tengo. En este caso son casas de alquiler, todo es configurable.

The screenshot shows a 'PIVOTE' view with the following details:

- VISTAS > PIVOTE** button.
- DESHACER | REHACER** buttons.
- Medidas** dropdown menu with options: Total, Vacía, Alquilada, Vendida.
- Insertar en hoja de cálculo** button.
- Etapa** column header.
- Precio** column header.
- Total** row: 1.200,00, 600,00, 900,00, 2.700,00.

On the left sidebar:

- Agrupación de columnas**: **Etapa**.
- Agrupar filas - primer nivel**.
- Medidas**: **Precio (App Alquiler Casas Nuevas)**.

Nuevo informe. En el nuevo informe puedo incluir todo lo que yo quiera. Puedo recuperar los datos que tenga la parte back, para incluirlos en mi informe. Le doy a guardar. Cerramos.

The screenshot shows the 'INFORMES' section with the following details:

- Nombre del informe**: App Alquiler Casas Nuevas Informe.
- Formato de papel**.
- Mostrar en el menú de impresión** checkbox checked.
- Recargar desde adjunto** checkbox unchecked.
- Limitar visibilidad a grupos**.
- EDITAR FUENTES** button.
- REESTABLECER INFORME** button.
- VISTA PREVIA DE IMPRESIÓN** button.

The main area displays a 'Company address block' component with the following description:

Company address block
Contains the company address.

Below it, there is a 'Lista' section with the following fields:

if: "address"
Dirección: dirección
Precio: 0,0 eur
Estado: estado

Sacar un listado: A modo lista. La selecciono. Y le digo que le **imprima** con el informe que he creado.

Alquiler Casas Nuevas App Alquier Casas Nuevas Configuración		
Nuevo	App Alquier Casas Nuevas	
		3 seleccionado x
<input checked="" type="checkbox"/> Descripción		App Alquier Casas Nuevas Informe
<input checked="" type="checkbox"/>	Casa 1	1.200,00 €
<input checked="" type="checkbox"/>	Casa 2	900,00 €
<input checked="" type="checkbox"/>	Casa 3	600,00 €
		2.700,00 €

Odoo Studio me permite personalizar todo lo que es de un módulo en completo.

Vista gráfico: incluir todos los parámetros que quiera incluir en el gráfico. Y los datos que yo quiera exportar. De modo que corresponde con estas vistas. Puedo ir marcando y seleccionando lo que yo quiera.

Odoo Studio puede llegar a mayor nivel toda la personalización que estoy realizando.

SISTEMA DE GESTIÓN EMPRESARIAL

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Python
comando pip
IDE PyCharm
palabras reservadas
print() e input()
operadores

¿Qué es Python?

Python fue creado a finales de los ochenta por Guido Van Rossum. A priori, se puede pensar que el nombre proviene de la serpiente pitón (del inglés *python*), pero no es así. Realmente, se debe a la afición que su creador tenía hacia los *Monty Python*, un grupo británico de humoristas con películas como “*La vida de Brian*” (1979).

Python es un lenguaje de programación:

- Interpretado.
- Multiparadigma.
- De alto nivel.
- Tipado dinámico y fuerte.

Características de Python

Python es un lenguaje de programación:

- Interpretado, es decir, que a diferencia de los compilados que requieren de un paso previo antes de ser ejecutado (la compilación), para convertir el lenguaje de alto nivel en código máquina, en los interpretados se va **convirtiendo a medida que se va ejecutando**.
- Multiparadigma, puesto que soporta **programación orientada a objetos, imperativa y funcional**.
- De alto nivel, ya que expresa los algoritmos de una manera **fácilmente comprensible** por el ser humano.
- Tipado dinámico y fuerte, puesto que la comprobación de la tipificación se hace durante la ejecución y además **no permite violaciones de los tipos de datos**, por ejemplo, no permite operar un carácter con un número. Si tenemos un número, las operaciones deben ser con números y lo mismo con caracteres.

El **núcleo del lenguaje es su gramática**, definida oficialmente en el siguiente vínculo:

<https://docs.python.org/3/reference/grammar.html>

A la hora de preguntarnos **por qué usar Python**, podemos llegar a las siguientes consideraciones:

- Es open source.
- Es multiplataforma (Mac, Windows y Linux).
- Es parecido a leer y escribir en inglés.

- Viene con una **librería estándar** (<https://docs.python.org/3/library/index.html>) muy completa que le permite **interaccionar con otros lenguajes**, bases de datos, etc.
- Dispone de un gran abanico de **librerías de terceros** (muchas empaquetadas en <https://pypi.org/>) para afrontar problemas como la **inteligencia artificial** (Keras, Tensorflow), **Big Data** (Pydoop), **Data Science** (Pandas, Numpy), **Testing** (Pytest, Robot), **web** (Django), **Scraping** (Urllib, selenium).
- Es el lenguaje con **mayor tasa de crecimiento** en los últimos años. Según el informe anual de GitHub, lo que ellos llaman el “Estado del Octoverso”, Python **superó a Java en 2019** y se colocó en segunda posición, justo por detrás de JavaScript.

Instalación de Python

Para instalar Python en Windows, bastará con descargarlo desde su web oficial (<https://www.python.org/downloads/>) y, seguidamente, ejecutarlo.

En cambio, en **Mac** ya existe una **versión preinstalada**, ya que **macOS lo usa** para su propio funcionamiento. No obstante, también **es posible instalar otra versión**, siendo los pasos idénticos que en Windows.

En **GNU/Linux**, al igual que pasaba en Mac, aquí también existe una **versión preinstalada**, porque lo usan también como **recurso propio**. Para **instalar otra versión**, lo más fácil es usar el **administrador de paquetes**.

```
sudo apt-get install python3
```

Como nota, indicar que se puede tener varias versiones de Python instalada en el mismo PC, no existe contraindicaciones sobre esto.

Una vez instalado, por ejemplo, en **Windows**, bastará con abrir '**símbolo del sistema**' y empezar a usar Python. El terminal tiene dos modos de trabajo:

1. Interactivo: las **instrucciones** que vayamos escribiendo, las irá ejecutando **una a una**. Para ello, deberemos ejecutar el **comando python (o python3 en Linux)**. Un mensaje con la versión de Python abierta y los '`>>>`' nos indicarán que estamos en Python y ya podremos realizar acciones desde el terminal:

Ejemplo de sesión de Python interactivo en Linux

```
Python 3.11.2 (main, Mar 13 2023, 12:18:29) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> █
```

Para salir de Python, bastará con ejecutar el comando `exit()` o `Ctrl + d`.

2. Ejecutando scripts: los scripts son archivos con **extensión ".py"** que se pueden editar con cualquier editor de textos, como, por ejemplo, Notepad, y en cuyo interior está el código implementado. Bastará con ejecutarlos usando el comando:

'python' + <nombre del archivo>

Creación y ejecución de script Python desde el terminal en Linux

```
marc@debian-12:~$ cat > script.py
print('Hola Mundo')
marc@debian-12:~$ python3 script.py
Hola Mundo
marc@debian-12:~$ █
```

Instalando Python en Windows

Describiremos cómo instalar Python en nuestro PC con un sistema operativo Windows:

Para ello, en primer lugar, deberemos entrar en la web oficial de Python, que es python.org, y pinchar en la pestaña de descargas, concretamente en All Release, porque desde aquí podremos ver todas las versiones de Python y descargarnos la que más nos interese. Están ordenadas por fecha de más moderna a más antigua.

En nuestro caso, nos vamos a descargar la versión más actual para Windows. Como nuestro sistema operativo es de 64-bit, podremos pinchar sobre Windows, que es nuestro sistema operativo, y aquí en la última versión que es la 3.8.3, podremos bajárnosla, la versión de 64-bit.

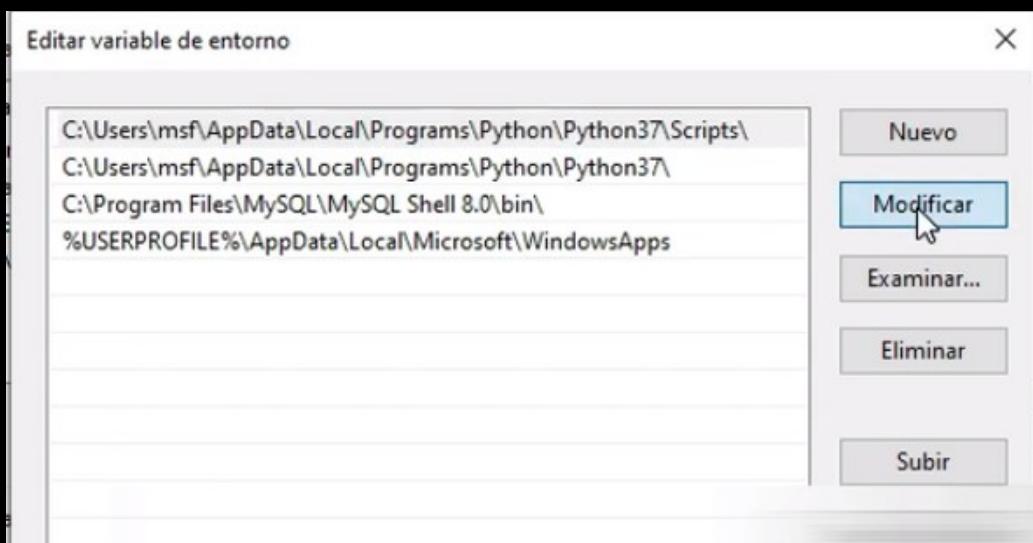
Una vez descargada, la abriremos y seguiremos los pasos del instalador. Podremos tener **diferentes versiones de Python instaladas** en el mismo PC. Esperamos que termine la instalación y una vez finalizado, cerraremos la ventana.

Tras esto, ya podremos abrir la consola de Windows y escribir Python. En este caso, nos ha abierto la versión 3.7.4, que la tendríamos también instalada en nuestro ordenador.

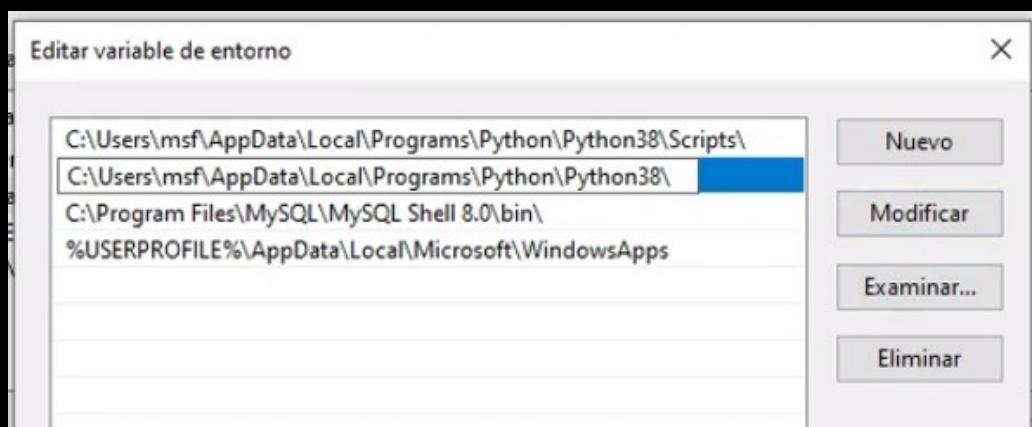
Si quisiéramos abrir la versión 3.8.3, es decir la última que hayamos instalado que es la que actualmente acabamos de abrir, (en vuestro caso será otra versión más reciente), para ello tendríamos que entrar a modificar las variables del entorno:

Entraríamos en las propiedades de este equipo → en configuraciones avanzadas del sistema → variables del entorno → path.

Aquí podemos ver que la versión que tenemos es la 3.7, tanto la versión en script como sin script.



Pues tendríamos que modificar y ponerle un 8 y aquí igual, porque como vemos en la misma ruta está tanto la versión 3.7 como la versión 3.8 y guardamos los cambios.



Tras guardar los cambios, tendremos que abrir el terminal de nuevo, reinicializarlo y ahora si ejecutamos el comando Python ya si vemos que es la versión 3.8.3 y desde aquí ya podremos realizar las operaciones oportunas.

IDE PyCharm

Usaremos la versión gratuita (**Community Edition**) del IDE PyCharm. Este está desarrollado por JetBrains y es multiplataforma. Para instalarlo, bastará con descargarlo de su **página oficial**:

<https://www.jetbrains.com/es-es/pycharm/>

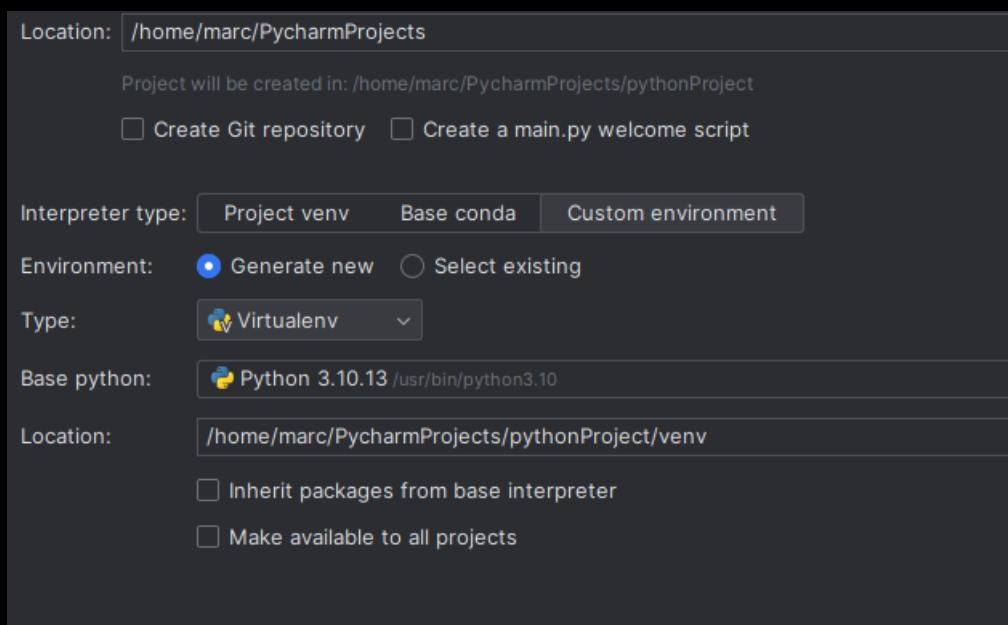
Este IDE nos permitirá, entre otras cosas:

- **completar código** con atajos de teclado,
- **navegar** por el código (por ejemplo, entre las clases),
- **refactorizarlo** (o limpiarlo),
- **detectar errores** o advertencias,
- usar **plugins** que permitirán, por ejemplo, interaccionar con otros lenguajes y frameworks (como Node JS),
- un **acceso a base de datos** más fácil,
- opción de **debugging**,
- etc.

Para comenzar a trabajar con PyCharm, crearemos un nuevo proyecto:

A la hora de crear un proyecto nuevo, PyCharm nos da la opción de seleccionar un **nuevo entorno** usando Virtualenv o un intérprete existente. La diferencia entre una y otra opción es que si se usa el entorno Virtualenv para el proyecto, todos los **paquetes que se agreguen, modifiquen o eliminén solo afectarán a ese proyecto**. En cambio, si se usa el intérprete existente, los cambios que se hagan **afectarán a todos los proyectos** que use dicho interprete.

Nuevo proyecto Python en PyCharm



En nuestros proyectos, por ahora, usaremos el entorno Virtualenv.

Podremos encontrar más información sobre la configuración del intérprete en el siguiente enlace:

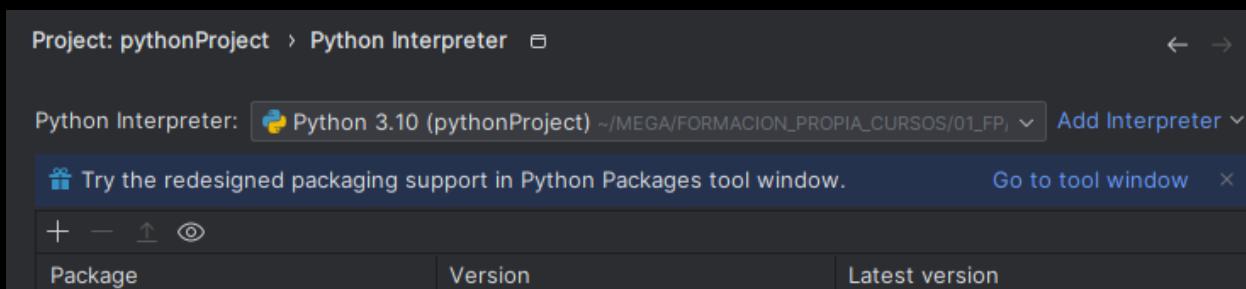
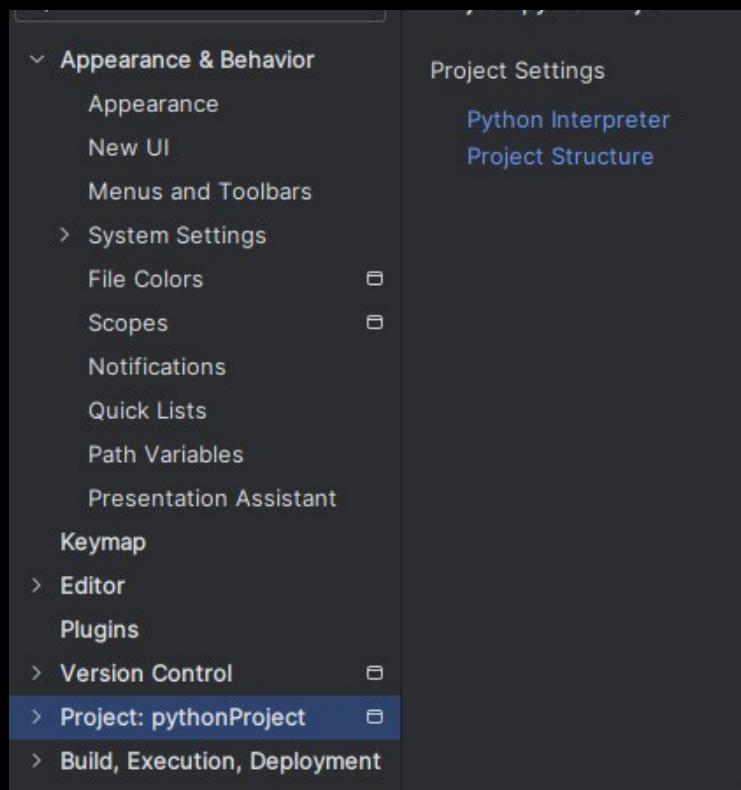
<https://www.jetbrains.com/help/pycharm/configuring-python-interpreter.html#add-existing-interpreter>

Hola mundo

Se deberá **crear el fichero Python** en nuestro proyecto. Para ello, bastará con pulsar el botón derecho sobre la carpeta del proyecto, y seguidamente, en 'nuevo' y 'archivo Python'. Esto nos creará un archivo '.py'.

A continuación, habrá que **configurar el intérprete** para que PyCharm sepa dónde está el archivo a ejecutar y con qué interprete ejecutarlo. Para ello, arriba a la derecha, junto al ícono de 'play', se pulsará en '**editar configuración**' e informaremos de la ruta del archivo Python y el intérprete.

Configurando el intérprete



Con esto, ya estaremos en disposición de crear y ejecutar nuestra primera práctica. Para ello, usaremos el comando para que nos muestre por pantalla “Hola mundo”:

```
print('Hola Mundo')
```

Finalmente, para ver el resultado, pulsaremos sobre el icono de ‘play’ , que nos mostrará el resultado en la consola de PyCharm:

Visualizando el resultado de la ejecución

```
Hola Mundo  
Process finished with exit code 0
```

El comando pip

Es el acrónimo de '**Pip Instalador de Paquetes**' o 'Pip Instalador de Python', y es un **sistema de gestión de paquetes** usado para **instalar paquetes escritos en Python**.

En el siguiente enlace, se podrá encontrar toda la información referente a pip:
<https://pip.pypa.io/en/stable/>

De esta forma, por ejemplo, para instalar librerías de terceros, una vez que conozcamos el nombre de la librería, bastará con usar el comando pip en el terminal (fuera de Python) con la instrucción install.

Por ejemplo, para instalar la librería redis, bastará con escribir el siguiente comando en el terminal:

Instalando una librería con el comando pip

```
pip install redis
```

El uso de **librerías de terceros**, dada la gran penetración del lenguaje entre el sector del desarrollo, hace que Python se convierta en una de las **opciones más potentes y más completas** a la hora de afrontar **cualquier proyecto de programación**.

Palabras reservadas y comentarios

En Python, nos encontramos con **treinta y tres palabras reservadas**:

https://www.w3schools.com/python/python_ref_keywords.asp

and	A logical operator
as	To create an alias
assert	For debugging
break	To break out of a loop
class	To define a class
continue	To continue to the next iteration of a loop
def	To define a function
del	To delete an object
elif	Used in conditional statements, same as else if
else	Used in conditional statements
except	Used with exceptions, what to do when an exception occurs
False	Boolean value, result of comparison operations
finally	Used with exceptions, a block of code that will be executed no matter if there is an exception or not
for	To create a for loop
from	To import specific parts of a module
global	To declare a global variable
if	To make a conditional statement
import	To import a module
in	To check if a value is present in a list, tuple, etc.
is	To test if two variables are equal
lambda	To create an anonymous function
None	Represents a null value
nonlocal	To declare a non-local variable
not	A logical operator
or	A logical operator
pass	A null statement, a statement that will do nothing
raise	To raise an exception
return	To exit a function and return a value
True	Boolean value, result of comparison operations
try	To make a try...except statement
while	To create a while loop
with	Used to simplify exception handling
yield	To return a list of values from a generator

Con respecto a los comentarios, estos pueden ser mono-línea, usando `#`, o multi-línea, usando **tres comillas dobles ("")** al inicio y al final del comentario.

```
# Este es un comentario mono-línea
```

```
"""
```

```
Y este es un comentario multi-línea
```

```
"""
```

Tipos de datos

En primer lugar, indicar que, en todo momento, podremos **saber el tipo de dato** de una variable usando la **función type()**:

Código:

```
varObjStr = 'Hola'  
varObjInt = 0  
  
print(type(varObjStr))  
print(type(varObjInt))
```

Resultado en consola:

```
<class 'str'>  
<class 'int'>
```

Las variables y constantes pueden ser de tipo:

- Numéricas:
 - **Int** → número = 0
 - **Float** → número = 0.0
 - **Complex** → número 3+1j
- Cadenas: **str** → texto = “hola mundo”
- Booleano: **bool** → booleano = True (o False)
- Listas: **list** → lista = [“dato1”, “dato2”...]
- Tuplas: **tuple** → tupla = (“dato1”, “dato2”, ...)

La diferencia entre las listas y las tuplas:

En las tuplas, el **contenido no es mutable**, es decir, no se puede cambiar su valor. Además, debido a esto, las tuplas ocupan **menos memoria**.

Ambas (listas y tuplas) son **heterogéneas**, es decir, pueden tener, por ejemplo, números y cadenas mezclados en su interior.

- Rangos: **range** → rango = range(4)

- Diccionarios o json (almacenan clave y valor diccionario):

dic → diccionario = {"nombre" : "Miguel", "edad" : "25"}

- Set (se trata de una colección que no está ordenada ni indexada):

set → set = ("dato1", "dato2" ...)

Casting

Hay ocasiones en las que nos interesa que **una variable sea de un tipo concreto**, aunque estas se hayan **declarado de otro tipo**. Para ello, se usan las siguientes funciones:

- `int()`: construye un número entero a partir de un literal entero, un literal flotante (redondeando al número entero anterior) o un **literal de cadena** (**siempre que la cadena represente un número entero**).
- `float()`: construye un número flotante a partir de un literal entero, un literal flotante o un literal de cadena (**siempre que la cadena represente un flotante o un entero**). **Ojo**.
- `str()`: construye una cadena a partir de una amplia variedad de tipos de datos, incluidas **cadenas**, literales **enteros** y literales **flotantes**.

Castings

```
print("tipo str " + varStr)
print(type(int(varStr))) # casting a int desde String

print("tipo int " + str(varInt))
print(type(float(varInt))) # casting a float desde int

print("tipo float " + str(varFloat))
print(type(str(varFloat))) # casting a String desde float
```

salida consola

```
tipo str 2
<class 'int'>
```

```
tipo int 2
<class 'float'>
```

```
tipo float 2.0
<class 'str'>
```

Las funciones de print e input

Hablaremos de las funciones Input y Print. Para ello, haremos uso de un pequeño programa que hemos hecho en PyCharm, del cual iremos comentando y descomentando el código para realizar diferentes pruebas.

1. Función Input:

En primer lugar, hablaremos del Input, que se utiliza para leer la información que el usuario introduce por teclado. Pausa el programa hasta que el usuario pulsa la tecla ENTER y siempre devuelve un string. Por ejemplo, ejecutamos la práctica y asignamos la entrada del usuario a una variable de test, luego la imprimimos. Si el usuario ingresa "123" y presiona ENTER, imprimirá "123". Sin embargo, podemos solicitar información al usuario incluyendo un parámetro en el Input, como en el caso de pedir el nombre.

Ejemplo:

```
nombre = input("¿Cuál es tu nombre?")
print(nombre)
```

2. Trabajando con Números:

El Input siempre devuelve un string, por lo que si tratamos con números, necesitamos convertirlos. Si preguntamos la edad y operamos directamente, se producirá un error. Utilizamos un casting para convertir la variable a entero antes de operar.

Ejemplo:

```
edad = int(input("¿Cuál es su edad?"))
print(edad + 1)
```

3. Función Print:

La función Print se utiliza para imprimir por pantalla. Podemos utilizarla separando diferentes variables o textos por coma. Si algún elemento no es un string, **Python lo convierte automáticamente.**

Ejemplo:

```
nombre = "Miguel"  
apellido = "Sánchez"  
edad = 25  
print(nombre, apellido, edad)
```

4. Parámetros de la Función Print:

La función Print tiene parámetros como end y sep. Por ejemplo, podemos especificar un separador y un finalizador.

```
print("Hola", "Adiós", sep="...", end="---")
```

Salida en consola:

Hola...Adiós---

5. F-Strings:

Desde la versión 3.6, se introdujo el fstring. Se coloca una F antes del texto y permite incluir expresiones directamente dentro del string.

Ejemplo:

```
fruta = "manzana"  
precio = 2.0  
cantidad = 8
```

```
print(f"El coste de la {fruta} es {precio}. \nPrecio total de 8 {fruta}s: {precio * cantidad}")
```

Salida en consola:

El coste de la manzana es 2.0.
Precio total de 8 manzanas: 16.0

Variables

Las variables son como contenedores para almacenar valores. En Python, no hay un comando para declarar variables, estas **se crean en el momento en el que se le asigna un valor** y **pueden cambiar de tipo** en el transcurso del código, por ejemplo:

Variables en Python

```
# Tipado dinámico
var = 1 # aquí la variable es de tipo numérico
var = "Hola mundo" # aquí la variable pasa a ser de tipo cadena
```

Para **declarar variables**, hay que seguir las siguientes reglas:

- Deben **empezar** por letras o por '_', **no por números**.
- Solo pueden tener **letras, números y el símbolo '_'**.
- Python es case **sensitive**, es decir, sensible a las mayúsculas y minúsculas (por ejemplo, test es distinto de Test).

Constantes

Una constante es una variable que siempre **mantiene el mismo valor**. En Python, **no existen constantes como tal**, lo que se hace es declarar una **variable en mayúsculas** para indicar que es una constante, por ejemplo:

```
CONSTANTE = 3
```

Errores

En Python, podemos encontrarnos con errores:

- De sintaxis: cuando escribimos mal el código y el intérprete no lo entiende, por ejemplo, no cerrar un paréntesis.

Error de sintaxis

```
1 + ("z"  
     ^  
  
SyntaxError: '(' was never closed  
Process finished with exit code 1
```

- De ejecución: no tiene error de sintaxis, pero el intérprete no lo puede ejecutar, por ejemplo, sumar un número y un carácter.

Error de ejecución

```
1 + "z"  
TypeError: unsupported operand type(s) for +: 'int' and 'str'  
Process finished with exit code 1
```

Leer la información que nos ofrecen los errores nos ayudará a subsanarlos.

¡Ojo! `input()` siempre devuelve un string

Vamos a crear un programa que **sume dos operandos** que el usuario introduce por pantalla y luego **los multiplique**.

El programa preguntará por el Operando1: y el Operando 2:, y luego devolverá la suma de ambos y la multiplicación.

Tendremos que hacer uso de las funciones `input()` y `print()`.

En primer lugar, **guardaremos en las variables “op1” y “op2” los operandos** que el usuario informe por pantalla:

```
op1 = input("Operando 1: ")
op2 = input("Operando 2: ")
```

Y, a continuación, imprimimos por pantalla el **resultado de la suma y la multiplicación**:

```
print(op1 + op2)
print(op1 * op2)
```

Ahora, si ejecutamos el programa e introducimos los valores 2 y 3, los resultados esperados son 5 para la suma y 6 para la multiplicación:

En su lugar, nos ha devuelto 23 para la suma y para la multiplicación, un **error indicando que no se pueden multiplicar tipos cadenas**.

Y esto es debido a que la **función `input()` siempre devuelve un string**, y claro, el operador '+' sí funciona en los strings como concatenación, pero el '*', no.

Para poder operar con los operandos, debemos hacer un **casting a tipo int**. Por ello, para recoger el valor adecuadamente, podríamos implementar la siguiente instrucción:

Casting

```
op1 = int(input("Operando 1: "))
op2 = int(input("Operando 2: "))
```

Y, finalmente, este sería el resultado:

Operando 1: 1

Operando 2: 2

3

2

Operadores

En Python, podemos encontrarnos con:

Aritméticos:

para operaciones matemáticas.

Operador	Nombre	Ejemplo
----------	--------	---------

+ suma $x + y$

- resta $x - y$

* multiplicación $x * y$

/ división x / y

% resto $x \% y$

** exponente $x ** y$

// división con redondeo a la baja $x // y$

De asignación:

para asignar valores a variables.

OPERADOR - EJEMPLO - ES LO MISMO QUE

= $x = 1$ $x = 1$

+= $x += 1$ $x = x + 1$

-= $x -= 1$ $x = x - 1$

*= $x *= 1$ $x = x * 1$

/= $x /= 1$ $x = x / 1$

%= $x %= 1$ $x = x \% 1$

//= $x // 1$ $x = x // 1$

**= $x **= 1$ $x = x ** 1$

De comparación:

para comparar valores.

OPERADOR - NOMBRE - EJEMPLO

`==` igual que $x == y$

`!=` distinto de $x != y$

`>` o `>=` mayor/mayor o igual que $x > y$ o $x >= y$

`<` o `<=` menor/menor o igual que $x < y$ o $x <= y$

Lógicos:

para combinar sentencias condicionales.

OPERADOR - DESCRIPCIÓN - EJEMPLO

`and` Devuelve verdadero si ambos operandos son verdadero $x < 1 \text{ and } x < 3$

`or` Devuelve verdadero si al menos uno de los operandos es verdadero $x < 1 \text{ or } x < 3$

`not` devuelve lo contrario de lo que tenga el operando $\text{not}(x < 1)$

De identidad:

para comparar objetos, pero, concretamente, si son el **mismo objeto con la misma posición de memoria**, no si tienen el mismo valor.

OPERADOR - DESCRIPCIÓN - EJEMPLO

`is` Devuelve verdadero si ambas variables son el mismo objeto $x \text{ is } y$

`is not` Devuelve verdadero si ambas variables no son el mismo objeto $x \text{ is not } y$

De pertenencia: si un objeto está presente en otro.

OPERADOR - DESCRIPCIÓN - EJEMPLO

in Devuelve verdadero si un objeto está presente en otro. x in y

not in Devuelve verdadero si un objeto no está presente en otro. x not in y

bit a bit: se usan con números binarios.

OPERADOR - NOMBRE - DESCRIPCIÓN

& AND Devuelve 1 si ambos bits son 1

| OR Devuelve 1 si uno de los bits es 1

^ XOR Devuelve 1 si solo 1 de los bits es 1

~ NOT Invierte el bit

<< Rellena con ceros por la izquierda

>> Rellena con ceros por la derecha

Ejemplo de código Python usando input() y casting

Vamos a crear un programa en Python que pregunte por pantalla la edad del usuario y a continuación le muestre un mensaje indicándole los **años que le restan para llegar a los 100**, que será una constante.

El programa preguntará “¿Cuál es su edad?” y al introducirla, si ésta es, por ejemplo, 25, entonces mostrará el mensaje “Le faltan 75 años para alcanzar los 100.”.

Para dar solución al caso práctico inicial, tendremos que hacer uso de las funciones input() y print(). En primer lugar, crearemos la constante de la edad a alcanzar con valor 100:

Creación de la constante

```
EDAD_A_ALCANZAR = 100
```

Seguidamente, teniendo en cuenta que la función input() siempre devuelve un string, para poder operar con ella como un número, deberemos hacer un casting a tipo int. Por ello, para recoger el valor de la edad, podríamos implementar la siguiente instrucción:

Recoger valor de la edad

```
edad = int(input("¿Cuál es su edad?"))
```

A continuación, deberemos restarle a 100 dicha edad, guardando el resultado en la misma variable o en una nueva:

Resta

```
falta = EDAD_A_ALCANZAR - edad
```

Y, finalmente, mostrar el mensaje por pantalla:

Mensaje por pantalla

```
print("Le faltan", falta, "años para alcanzar los", EDAD_A_ALCANZAR)
```

Ahora, sí ejecutamos el programa:

Ejecución del programa

```
¿Cuál es su edad?46
```

```
Le faltan 54 años para alcanzar los 100
```

```
Process finished with exit code 0
```

SISTEMA DE GESTIÓN EMPRESARIAL

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Programando en Python

Indentación

IF

FOR y WHILE

Errores

Funciones, módulos y paquetes o librerías

Cadenas de textos

Ficheros

Orientación a objetos

Indentación y flujo secuencial

Indentación

La indentación en Python facilita tanto la comprensión como la lectura del código, pero ¿qué es la indentación?

Es un desfase de una o varias líneas de código hacia la derecha (mejor dicho varios espacios hacia la derecha), es decir, una especie de **sangrado**. Se trata de algo **muy importante** en Python y una de sus principales características.

Se usa para **agrupar sentencias** y así poder **diferenciar un bloque** de otro, ya que, en Python, **no hay terminadores** de sentencia (como el punto o el punto y coma que se usan en lenguajes como Java, C o C++) **ni llaves** para abrir y cerrar bloques. Esto evita que los programadores comentan los errores típicos que se comenten al olvidar estos delimitadores.

A partir de este punto, veremos algunos ejemplos de indentación, pero como adelanto, podremos analizar el siguiente:

Ejemplo de indentación

```
if <condición>:  
    instrucción si es verdadera la condición  
    Nueva instrucción no indentada # fuera del bloque condicional
```

Un IDE como **PyCharm** realiza el sangrado o **indentación automáticamente**, mientras que si nos decantamos por usar un editor de textos, como puede ser el bloc de notas, deberemos realizarlo nosotros manualmente haciendo uso de espacios o tabuladores.

Flujo de programación secuencial

En este punto, veremos la sentencia condicional IF, que se basa en que si se cumple una condición, realiza una instrucción concreta.

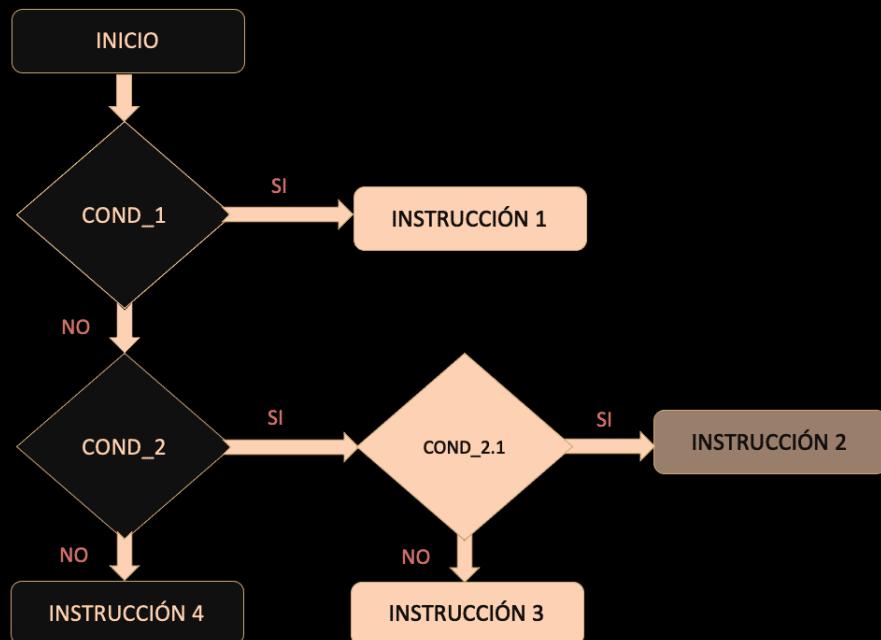
En Python, se declara así:

Declaración IF

```
if <condición 1>:
    instrucción 1: se cumple la condición 1
elif < condición 2>:
    if <condición 2.1>:
        instrucción 2: se cumplen las condiciones 2 y 2.1
    else:
        instrucción 3: se cumple la condición 2, pero no la 2.1
else:
    instrucción 4: no se cumplen las condiciones 1 ni 2
```

Correspondiéndose con el siguiente diagrama de flujo:

Diagrama de flujo del ejemplo de la sentencia condicional



Pudiéndose usar un formato acortado para el if y para el if-else:

Formato reducido IF

Con IF:

```
if a > b: print(f'{a} es mayor que {b}')
```

Con IF ... ELSE:

```
print(a) if a > b else print(b)
```

Flujo de programación iterativo y range

Python dispone de dos comandos **iterativos**: WHILE y FOR, y una función para generar **progresiones aritméticas**: RANGE.

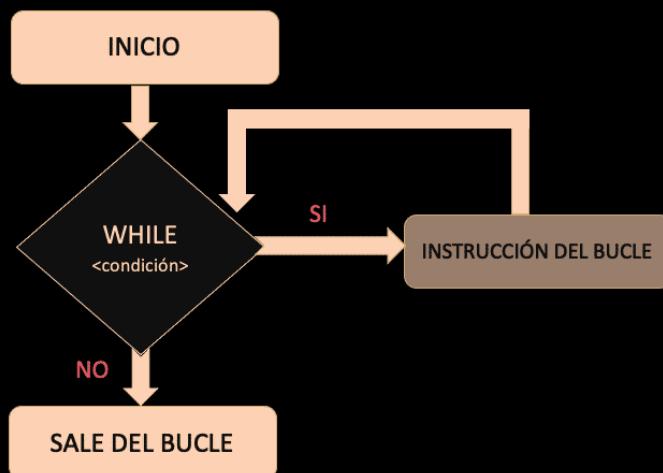
El bucle WHILE

Este bucle repite la/s instrucción/es que se encuentren en su interior mientras se cumpla la condición. En Python, se declara de la siguiente forma:

Declaración while

While <condición>:
instrucción/es

Diagrama de flujo del bucle while



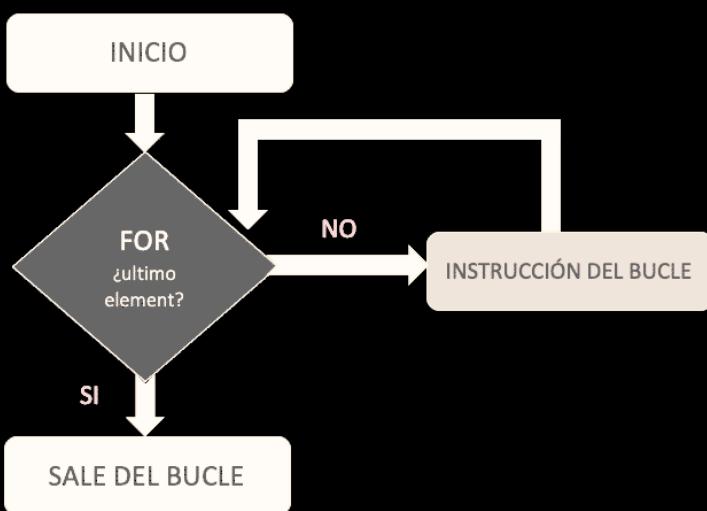
El bucle FOR

A diferencia de otros lenguajes, en Python, este bucle **se usa solo para iterar sobre secuencias** (lista, tupla, diccionario, cadena...) en el **orden en el que aparecen** los elementos en la secuencia. En Python, se declara así:

Declaración for

For <elemento> in <secuencia>:
Instrucción/es

Diagrama de flujo del bucle for



Para los bucles **WHILE** y **FOR**, existen las siguientes sentencias:

- Break: **para la iteración**, incluso si la condición es verdadera.
- Continue: para la iteración actual del bucle y **pasar a la siguiente**.

La función range

La función range() genera listas con progresiones aritméticas, muy útil para iterar con bucles.

Por ejemplo:

Range(3) generaría → [0, 1, 2].

Esta función **empieza en 0 por defecto** y va **incrementando** los valores **de 1 en 1**.

Pero podemos hacer que empiece en otro número y se incremente de otra forma.

Por ejemplo:

Range(5, 21, 5) generaría → [5, 10, 15, 20]

empieza en 5, termina en 21 y se incrementa de 5 en 5.

Control de errores

Cuando se produce un error en Python, este detiene la ejecución del programa y muestra el error. Pero estos son errores del estándar de Python, que pueden llegar a ser difíciles de entender por los usuarios finales. Por ello, Python nos permite **manejar el control de errores** con las siguientes instrucciones:

- **Try:**

Se usa al inicio del bloque para indicar que en ese bloque se estarán capturando errores.

- **Except:**

Bloque en el que se **maneja el error**. Se pueden definir tantos bloques except como necesitemos, indicando el tipo de error que se controlará. (Nota: como no existen llaves en Python, con Except se indica que empieza el manejo del error en caso de que suceda, similar al catch en Java).

Con el comando as, se nos permitirá **capturar el objeto** de excepción.

- **Finally:**

Bloque que **se ejecutará siempre** al final, tanto si hay errores como si no.

Además, Python nos permite hacer uso de la instrucción else para definir un bloque de código **cuando no se produzcan errores**.

También, con el comando raise, Python nos da la opción de **lanzar excepciones** indicando, incluso, el tipo de excepción si fuese necesario.

Un ejemplo completo sería el siguiente:

Ejemplo de control de errores

```
try:
```

```
    #instrucciones (10/0) ("a"+2) (2+2)
```

```
except ArithmeticError as error: #captura errores aritméticos
```

```
    print("Se ha producido el siguiente error aritmético:", str(error))
```

```
except Exception as error: #captura el resto de errores
```

```
    print("Se ha producido el siguiente error:", str(error))
```

```
else: #entra si no se producen errores
```

```
    print("No se han producido errores")
```

```
finally: #entra siempre
```

```
    print("bloque finally, se ejecuta siempre")
```

De esta forma:

- Si la instrucción fuese `10/0`, devolvería el mensaje de error aritmético y seguidamente el del `finally`.
- Si la instrucción fuese `"a" + 2`, devolvería el mensaje de error del exception y luego el del `finally`.
- Si la instrucción fuese `2+2`, devolvería `4`, el mensaje indicando que no se han producido errores y el del `finally`.

Ejemplos de programación Python

Sentencia condicional IF, los bucles WHILE y FOR con RANGE

Hablaremos de algunas de las sentencias que hemos visto hasta ahora en este tema. Empezaremos hablando de la **sentencia `if`**.

Por ejemplo, en este caso tenemos dos variables `x` e `y`.

Si `x` es mayor que `y`, pintaremos "x mayor que y".

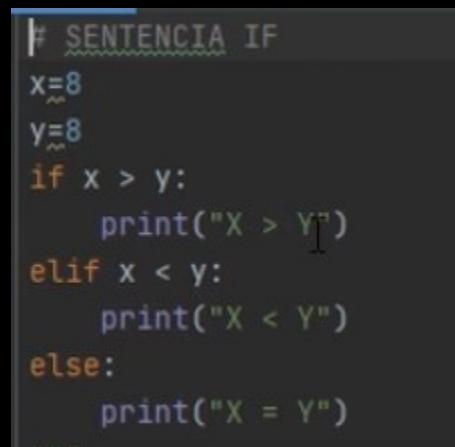
Si `x` es menor que `y`, pintaremos "x menor que y".

Y si no, pintaremos "x igual a y".

En este caso, por ejemplo, si `x` valiese 7, pintaría "x menor que y", porque es menor que 8.

Si fuese mayor, pintaría "x mayor que y".

Y si fueran iguales, entraríamos por el `else` y pintaría "x igual a y".



```
# SENTENCIA IF
x=8
y=8
if x > y:
    print("X > Y")
elif x < y:
    print("X < Y")
else:
    print("X = Y")
```

Esta misma sentencia IF se puede usar de una forma más corta, como se muestra a continuación:

Pintaremos "X > Y", si `x` es mayor que `y`.

Si no, pintaremos "X < Y", si `x` es menor que `y`.

Y si no, pintaremos "X = Y".

Podremos ir encadenando tantos `else` como necesitemos.

Por ejemplo, en este caso, si `x` es menor que `y` y ejecutamos, nos pinta que "X < Y".

Si `x` es mayor que `y` y ejecutamos, nos pinta que "X > Y".

Y si `x` fuese igual que `y` y ejecutamos, nos pinta que "X = Y".

Con esto ya hemos visto la sentencia `if`.

```
# SENTENCIA IF ACORTADA
x=5
y=5
print("X > Y") if x > y else print("X < Y") if x < y else print("X = Y")
```

Ahora, por ejemplo, queremos el **bucle `while`**.

En este caso, lo que haremos será recorrer un bucle incrementando en 1 la variable `x` mientras `x` sea menor que 20.

Si el resto dividido entre 2 es igual a 0, eso significa que el número es par. Entonces, no pintaremos nada.

Si `x` es igual a 15, terminaremos el bucle y, en otros casos, iremos pintando el valor de `x`. Por ejemplo, en el primer caso, `x` valdrá 0, que el resto es 0. Entonces, no pintará nada.

Con el `continue`, volverá arriba. `x` le incrementa 1. Entonces, `x` valdrá 1. El resto dividido entre 2 no es igual a 0. Tampoco es el valor 15. Entonces, lo pintará e incrementará 1. Ahora volvemos al 2, que es par, por lo que incrementará el valor de `x`, continuará y volverá aquí arriba. `x` ahora es 3, no es par, no es 15, lo pintará. Y así hasta el valor 15, que hará que salga del bucle igual. En este caso, pintaremos los números entre 1 y 15 que no sean pares.

```
# BUCLE WHILE
x = 0
while x < 20:
    if x % 2 == 0: # si el resto de dividirlo entre 2 es 0 -> es par
        x += 1
        continue
    if x == 15:
        break
    print(x)
    x += 1
```

Ahora hablaremos sobre el **bucle `for`**.

El bucle `for` va a recorrer una lista, y en este caso es una lista de deportes, de cadenas de texto de deportes. Por cada elemento de la lista de deportes, que llamaremos `deporte`, si el **primer carácter es una 'f'**, **continúa**. Es decir, que no hace nada. Vuelve al siguiente elemento de la lista. Si el deporte es "equitación", termina de recorrer el bucle, y si no, pinta. En este caso, entrará el primer elemento, la 'f', como el primer carácter es la 'f', pues el "fútbol", el primer carácter es la 'f', pues continúa y no lo pinta. El siguiente "baloncesto" no es la 'f' ni es "equitación", entonces lo pinta. "Fórmula 1", si es la 'f', continúa. "Tenis" no es la 'f' ni es "equitación", entonces lo pinta. Y en el siguiente elemento, que si es "equitación", no es 'f' pero es "equitación", entonces **hace el `break` que sale del bucle**. Pintará entonces "baloncesto" y "tenis", como podemos apreciar en este ejemplo.

```
# BUCLE FOR
deportes = ["futbol", "baloncesto", "formula 1", "tenis",
            "equitación", "ciclismo"]
for deporte in deportes:
    if deporte[0] == "f":
        continue
    elif deporte == "equitación":
        break
    print(deporte)
```

A continuación, hablaremos del uso del `range` .

En este caso, por ejemplo, vamos a recorrer un bucle para una variable `x` :

en el rango **de 5 hasta 45**,

de 10 en 10, y vamos a pintar `x` .

Es decir, empezaría por el valor 5, luego le incrementaría 10, y así hasta que el valor llegue a 45.

```
# RANGE
for x in range(5,45,10):
    print(x)
```

Ejemplo: Hallar el número entero mayor y menor de una lista

Necesitamos implementar un programa que devuelva el **número entero mayor y menor de una lista heterogénea**, es decir, podrá tener números enteros, reales, caracteres, cadenas, etc.

Por ejemplo, para una lista como la que sigue: lista = [1, 2, "árbol", 5, 1.5, 6, "a"]

el programa devolverá: Menor: 1 / Mayor: 6

Para ello, en primer lugar, crearemos las variables 'menor' y 'mayor' y las inicializaremos a vacío (**None**).

A continuación, recorreremos la lista, y por cada elemento, comprobaremos si es un entero. En caso afirmativo, chequearemos si dicho entero es menor que nuestra variable menor; entonces, la variable menor tomará ese valor.

A continuación, haremos lo mismo para la variable mayor, es decir, si el elemento es mayor que la variable mayor, entonces, la variable mayor tomará dicho valor. De esta forma, una vez que se termine de recorrer la lista, tendremos en 'menor' el valor menor y en 'mayor' el valor mayor. Pero ¿qué ocurrirá en la primera iteración del bucle cuando compare el primer elemento entero con la variable 'menor' si esta tiene un valor None?

El programa dará el siguiente error:

Error que se da al comparar un tipo 'int' con un 'NoneType'

TypeError: '<' not supported between instances of 'int' and 'NoneType'

El motivo es que **no se puede comparar un tipo 'int' con un 'NoneType'**. Por ello, antes de realizar las comparaciones, **si las variables 'menor' y 'mayor' están vacías**, les **daremos el valor** del primer elemento entero que nos encontremos, puesto que en ese momento será el menor y el mayor a la vez en la lista.

De esta forma, el código quedará como sigue:

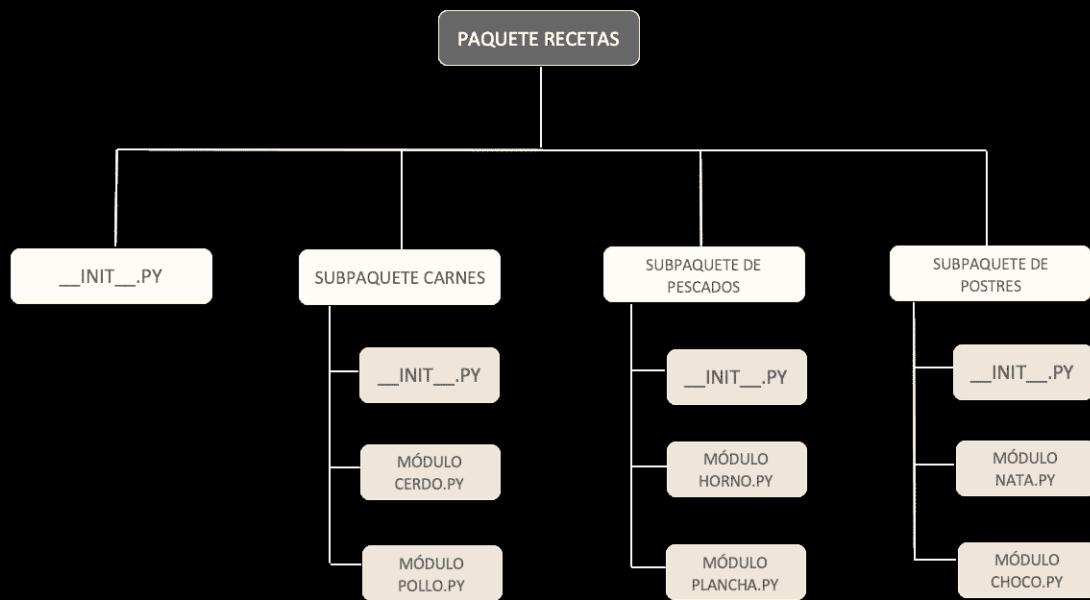
```
menor = None #reinicializamos la menor
mayor = None #reinicializamos la mayor
lista = [1, 2, "árbol", 5, 1.5, 6, "a"]
for x in lista:
    if type(x) is int:
        if menor == None and mayor == None: #es la primera iteración con un entero
            menor = mayor = x
        if x < menor:
            menor = x
        elif x > mayor:
```

```
mayor = x
print ("Menor:", menor)
print ("Mayor:", mayor)
```

Estructura del código: Funciones

Estructura del código

Estructura del código en Python



Estructurando el código

Para estructurar el código y hacerlo más potente, legible, reutilizable y portátil Python ofrece las funciones, los **módulos** y los **paquetes** o librerías.

Esto nos permitirá:

Crear código -> empaquetarlo -> distribuirlo -> reutilizarlo.

Para comprender mejor los tres conceptos, de los que hablaremos a continuación, haremos una analogía con las recetas de cocina. De esta forma:

Las **funciones** vendrían a ser las recetas en sí, por ejemplo, la **receta** de una tarta de chocolate.

Los **módulos** los **libros de recetas**, por ejemplo, un libro de receta de postres.

Los **paquetes** (o librerías) estanterías llenas de libros de recetas, por ejemplos, libros de recetas de postres de diferentes autores con diferentes recetas.

Funciones: 'Las recetas'

Son como cajas negras que **realizan una serie de instrucciones cuando son llamadas**, de forma que al usarlas, solo nos preocupará saber qué hacen, pero no cómo lo hacen internamente con código Python. Suelen tener un nombre característico de la acción que realiza y **pueden recoger parámetros y devolver valores**.

Un ejemplo es la función `print(value1, value2, ..., sep=' ', end='\n...')`, cuya finalidad es imprimir por pantalla. Además, tiene los parámetros (opcionales) 'sep', 'end', etc., de los que ya hemos hablado.

Para **crear** una función, basta con usar:

Creación de una función

```
def<nombre_función> (parámetros si fuese necesario):  
    <implementación>  
    return <valor> opcional: si devolviese algún/os valor/es
```

Y para **llamarla**, basta con usar su nombre:

Llamar a una función

<nombre_función>(parámetros si tuviese)

Un posible ejemplo de declaración y uso de función sería el siguiente, dando como resultado: 7 y 12:

Ejemplo declaración y uso de función

```
def sum_mul (num1, num2):  
    return num1 + num2, num1 * num2  
sum, mul = sum_mul(3,4)  
print(sum)  
print(mul)
```

***Nota:** cuidado con las variables globales y locales. Las variables que se definan **dentro de la función** solo serán visibles ahí dentro. Si queremos **modificar dentro de la función una variable global**, hay que usar la **etiqueta 'global'**.

Estructura del código: Módulos y librerías

Módulos: 'Los libros de recetas'

Son ficheros que **contienen las funciones en código Python** y se denominan con **<nombre_modulo>.py**.

Antes de usarlo, debemos tener instalados el módulo, y para **comprobar los módulos** que tenemos instalados, bastará con ejecutar el comando `help('modules')` dentro de la consola de Python. Este dispone de muchos módulos estándar consultables desde aquí.

Para instalar un módulo, se puede hacer con el comando:

pip + install + <nombre del módulo>

Una vez creado o instalado el módulo, ya podemos **usarlo** haciendo uso de la sentencia:

import <módulo>

Por ejemplo, podremos crear el módulo 'holamundo.py' (extensión .py) con varias funciones en su interior entre las que se encuentra:

```
def holamundo():
    print("hola mundo")
```

A continuación, en nuestro programa, podremos importarlo y usar dicha función con la instrucción `def holamundo()`:

Importar holamundo

```
import holamundo
holamundo.holamundo()
```

Otra forma de importarlo sería usando el atributo from, así no tendremos que indicar el módulo al usar la función:

Importar con from

```
from holamundo import holamundo  
holamundo
```

Los paquetes o librerías: 'Las estanterías'

Los paquetes no son más que una carpeta que contiene varios módulos (archivos '*.py') y/o paquetes. Se caracterizan por tener un [archivo de inicialización '`_init__.py`'](#) (el contenido **suele estar vacío**). Python dispone de una librería estándar que le otorga al lenguaje multitud de funcionalidades. Consultala aquí: <https://docs.python.org/3/library/index.html>.

Pero, además, existen una gran cantidad de librerías de terceros fácilmente instalables: <https://pypi.org/>.

Con el **comando pip**, podremos:

- **instalar paquetes** (`pip install <paquete>`),
- **instalar una versión** concreta (`pip install -Iv <paquete>==<versión>`),
- **actualizarlo** (`pip install -U <paquete>==<versión>`),
- **desinstalarlo** (`pip uninstall <paquete>`)
- `pyy listar` todos (`pip list`).

Para usar un paquete, bastará con hacer un `import <paquete>` en nuestro proyecto y podremos hacer uso de todos sus módulos y funciones. Si quisieramos importar un módulo concreto, podríamos hacer:

```
import <paquete>.<módulo>
```

Ficheros y cadenas de texto

En Python, disponemos de funciones que nos permitirán crear, abrir, modificar, cerrar, eliminar y tratar ficheros.

Apertura de ficheros

El comando Open tiene dos parámetros:

- el nombre del fichero
- y el modo, que puede ser:
 - r: en modo lectura.
 - a: en modo **agregar o crear** si no existe.
 - w: en modo escritura.
 - x: modo **creación**.

```
open("<nombre_fichero>","<modo>")
```

Adicionalmente, se puede especificar si se desea tratar en modo:

- **texto** ("t" por defecto),
- o **binario** ("b").

De esta forma, para abrir un fichero en **modo escritura binaria**, podríamos usar:

```
f = open("archivo.txt", "wb")
```

Si no se especifica nada en <modo> **por defecto**, lo abre en modo **texto y lectura**.

Cierre de ficheros

Es una buena costumbre cerrar los ficheros cuando ya no se están tratando. Para ello, se usa la función close().

Continuando con el ejemplo anterior:

```
f.close()
```

Lectura de ficheros

Una vez abierto, podremos leer su contenido haciendo uso de la función `read()` (para leer todo su contenido) o `readline()` (para leer su contenido por líneas). Siguiendo el ejemplo:

- `print(f.read())`: imprimirá el **contenido** del archivo.
- `print(f.readline())`: imprimirá la **primera línea** del archivo.

Escritura de ficheros

Una vez abierto , podremos escribir en el mismo usando la instrucción `write()`. Continuando con el ejemplo anterior:

```
f.write("texto nuevo")
```

- Si lo abrimos con el 'w', se **sobrescribirá** el contenido del fichero;
- si lo abrimos con 'a', se **agregará**.

Otras funciones con ficheros

Existen muchas más funciones de las que destacaríamos:

- `tell()` que devuelve en qué **posición del archivo** estamos.
- `seek()` para **irnos a una posición** concreta del archivo.
- `flush()` para que **se realice** todo lo que tiene en **buffer**.

Trabajando con cadenas de texto

Analizaremos, con ejemplos en PyCharm, el uso de las cadenas de texto en Python.

Veremos cómo pueden ser declaradas tanto con comillas simples ('') como con dobles (""), el uso de "\ y "\n", las subcadenas, y algunas funciones.

Analizaremos el uso de las cadenas de texto en Python, con ejemplos.

Para empezar, comentar que Python permite tanto el uso de la **comilla simple** como la **comilla doble** para generar las cadenas de texto. De esta forma, la variable x y la variable y, una con comilla simple y otra con comilla doble, imprimirán el mismo valor.

```
# comillas simples y dobles
x = 'Hola mundo'
y = "Hola mundo"
```

```
print(x)
print(y)
```

Esto también se extraña a los textos de varias líneas. Por ejemplo, podemos usarlos con **cadenas simples o con cadenas dobles**. En este ejemplo, las líneas se imprimen con un retorno de carro o un intro al final.

```
# varias Lineas de texto
x = ""esto es una cadena
de texto de
varias Lineas"""
y = """esto es una cadena
de texto de
varias Lineas"""
print(x)
print(y)
```

Si no quisiéramos que se imprimiese ese retorno de carro, podríamos hacer uso de la **barra invertida**. En este caso, la barra invertida nos indica que **todo es una misma línea**. Si imprimimos el mismo texto con las barras estas al final, podemos ver que se imprime todo en una misma línea.

```
# la\
x = ""esto es una cadena \
de texto de \
varias Lineas"""
print(x)
```

resultado:

```
esto es una cadena de texto de varias Lineas
```

Si quisiéramos, de alguna forma, que todo lo escribiéramos en una línea, pero se imprimiese un intro, el carácter para el **retorno de carro es el "\n"**. En este caso, aunque todo esté en una misma línea, al imprimirlo, podremos ver que a partir del "\n" se imprime en otra línea.

```
# la \n
x = "esto es una Linea \nesto es otra"
print(x)
```

La concatenación de cadenas de caracteres se puede hacer de varias formas, pero la más fácil es usando el **carácter** "+".

En este caso, tenemos x con "hola " e y con "mundo", y al imprimir la concatenación se imprime "hola mundo".

```
# concatenación
x = "hola "
y = "mundo"
```

Pasamos a hablar del uso de la función `len()`, que es la **longitud**.

En este caso, tenemos una cadena que es "tamaño de la cadena", que si queremos imprimir la longitud de esa cadena, podemos hacer uso de la función `len()`. En este caso, nos indica que la cadena tiene 19 caracteres.

```
# función len()
x = "tamaño de la cadena"
print(len(x))
```

Ahora pasaremos a comentar el uso de las cadenas de texto como array.

Al fin y al cabo, un string en Python no deja de ser como un array, que ya conocemos de otros lenguajes. Por ello, podemos imprimir:

```
x = "0123456789"
```

- el **primer carácter** de la cadena con esta instrucción:

```
print(x[0])
```

resultado: 0

- los caracteres comprendidos entre la posición 8 y 9 de esta cadena:

```
print(x[7:9])
```

resultado: 78

- los caracteres comprendidos entre el inicio y la posición 7:

```
print(x[:7])
```

resultado: 0123456

- los comprendidos entre la posición 7 y el fin de la cadena:

```
print(x[7:])
```

resultado: 789

- los comprendidos entre el final menos 7, o sea, 7 caracteres para que llegue al final y el final:

```
print(x[-7:])
```

resultado: 3456789

- o en este último caso, los caracteres que están comprendidos desde la posición 3 hasta el final menos 1 de esta cadena.

```
print(x[3:-1])
```

resultado: 345678

Podremos decirle que nos imprima la cadena "hola mundo" todo en minúscula:

```
# imprime todo en minúsculas:  
x = "Hola Mundo"  
print(x.lower())
```

O todo en mayúscula:

```
# imprime todo en minúsculas:  
print(x.upper())
```

También podremos reemplazar caracteres o palabras de un string por otros.

En este caso, imprimiremos el resultado de reemplazar "hola" por "adiós". Y ahora tenemos "adiós mundo", pero en este caso solamente lo hemos hecho a la hora de imprimir:

```
# reemplazo de caracteres:  
x = "Hola mundo"  
print(x.replace('Hola', 'Adiós'))
```

Si intentamos hacer uso de la función `find()` para buscar la palabra "adiós" en la variable `x` nos va a decir que no está, nos va a **devolver menos 1**, porque realmente no lo hemos reemplazado, hemos imprimido el reemplazo:

```
print(x.replace('Hola', 'Adiós'))
```

Si imprimimos la `x` solamente, vemos que se imprime "hola mundo". Este "adiós mundo" es la impresión del reemplazo y esta `x` es la variable que realmente no ha sido cambiada.

```
print(x)
```

resultado:

-1

Hola mundo

Expicaremos el uso de la función `split()`, la cual nos **divide** nuestra **cadena** en **elementos de una lista**. Podemos apreciar que se imprime por un lado "hola" y por otro "mundo" como dos objetos en una lista.

```
# función split():  
x = "Hola mundo"  
y = x.split()  
print(y)
```

resultado: ['Hola', 'mundo']

Ejemplo de conversión de decimal a binario usando ficheros

Necesitamos crear un programa en Python que lea un fichero denominado ‘decimal.txt’ que contiene un número decimal y que lo convierta a binario y escriba el resultado en otro fichero denominado ‘binario.txt’.

En primer lugar, crearemos una función para convertir el decimal que se le pasa por parámetro a binario (sin hacer uso de la función bin()). A continuación, en el programa principal, abriremos el fichero decimal, leeremos su contenido y lo cerraremos. Seguidamente, llamaremos a la función que hemos creado para convertir el decimal, abriremos el fichero de salida, escribiremos el resultado en él y lo cerraremos.

La función la denominaremos ConvertirBin, tendrá un parámetro de entrada denominado ‘decimal’ y devolverá una cadena de unos y ceros.

Tendrá la siguiente forma:

```
def ConvertirBin(decimal):
    binario = ""
    # mientras la división entera del decimal entre 2 no sea 0
    while decimal // 2 != 0:
        # concatena el resto de dividirlo entre 2 al resultado por la izquierda
        binario = str(decimal % 2) + binario
        # volcar en el decimal el resultado de la división entera entre 2
        decimal = decimal // 2
    return str(decimal) + binario
```

A continuación, abriremos el fichero, lo leeremos y lo cerraremos:

```
fich = open("decimal.txt", "r")
dec = fich.read()
fich.close()
```

Después, llamaremos a la función que hemos creado para convertir el decimal, ojo, pero el que hemos leído está en formato str, deberemos hacerle un **casting a int** al pasarlo a la función:

Llamada a la función para convertir a binario

```
bin = ConvertirBin(int(dec))
```

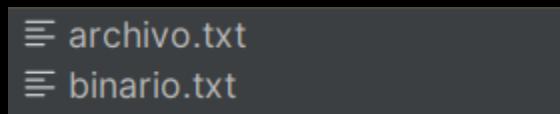
Finalmente, **abriremos el fichero de salida, escribiremos** el resultado en él y lo cerraremos.

Creación, escritura y cierre del fichero resultante

```
fich = open("binario.txt", "w")
fich.write(bin)
fich.close()
```

De esta forma, para el siguiente fichero: decimal.txt (Fichero de lectura)

El programa creará el siguiente resultado: binario.txt (Fichero de escritura)



The image shows a dark-themed file manager interface. At the top, there is a header bar with a search field containing the text 'binario.txt'. Below the header, there is a list of files. The first file listed is 'archivo.txt', followed by 'binario.txt'. Both files have a small icon to their left, which typically represents a text file.

Orientado a objetos

Python es un lenguaje de programación orientado a objetos. De hecho, **casi todo en Python es un objeto con sus propiedades y métodos**.

Crear una clase

Bastará con usar la **sentencia class**, definir su **método `_init_()`** (esto es recomendable), que será el constructor que se llamará automáticamente al crear un objeto de esa clase y definir los métodos de la clase:

Definir clase y métodos

```
class nombre_clase:  
    def __init__(self, propiedad1, propiedad2):  
        self.propiedad1 = propiedad1  
        self.propiedad2 = propiedad2  
  
    def nombre_metodo(self, parámetro1, parámetro2):  
        # acciones del método
```

Siendo `self` un parámetro que hace **referencia a la instancia actual de la clase** que se está llamando, y se deberá usar como **primer parámetro de todos los métodos de la clase** (se puede **nombrar como queramos**)

De esta forma, un ejemplo de creación de una clase podría ser:

Clase DatosUsuario

```
class DatosUsuario:  
    def __init__(self, nombre, edad):  
        self.nombre = nombre  
        self.edad = edad  
  
    def escribir_datos(self):  
        print(f"El nombre es: {self.nombre} y su edad es: {self.edad}")
```

Crear un objeto de esa clase

Bastará con **crear una variable llamando a la clase**:

Crear objeto

Objeto = <nombre_clase>(<propiedad1>, <propiedad2>...)

Siguiendo el ejemplo anterior:

Creación del objeto usuario de la clase DatosUsuario

usuario = **DatosUsuario("José", 46)**

Llamar a un método de una clase

Una vez creada la clase y el objeto de dicha clase, solo será necesario llamar al método poniendo un '.' entre el objeto y el método:

Llamada al método

Objeto.método(<parámetro1>, <parámetro2>...)

Siguiendo el ejemplo anterior, la llamada al siguiente método imprimirá el siguiente resultado:

Llamada al método escribir_datos de la clase DatosUsuario

usuario.**escribir_datos()**

Resultado: El nombre es: José y su edad es: 46

Ejemplo de uso de librerías de terceros

Necesitamos implementar un programa en Python que pinte por pantalla figuras geométricas dependiendo del número de lados que introduzca el usuario por pantalla.

Hay que tener en cuenta que cada figura geométrica de x lados comparte la propiedad que la suma de sus lados es 360° . Así, sabemos que, por ejemplo, un triángulo deberá girar en cada esquina 120° ($360/3$), un cuadrado deberá girar en cada esquina 90° ($360/4$).

Algo que, a priori, parece tan complejo, es muy fácil haciendo uso de librerías. Si echamos un vistazo a la librería turtle aquí, podemos apreciar que esta tiene las siguientes funciones:

- Screen(): para pintar la pantalla.
- Turtle(): para crear el objeto Turtle.
- Forward(x): mueve a la tortuga la **distancia x que se le indique**.
- Left(x): **gira** la tortuga hacia la izquierda **x ángulos**.

Nota: Para solucionar el error ModuleNotFoundError: No module named 'turtle', podría funcionar:

https://www.youtube.com/watch?v=0x_MEKr0OJQ

Si no se puede instalar directamente desde la terminal:

`pip install turtle`

o

`apt install python3-tkinter.`

En este caso, habría que usar la terminal para el código, y funciona.

Ya solo nos quedará **implementar el programa** con los datos específicos. Para ello, en primer lugar, **importaremos la librería** turtle:

```
import turtle
```

A continuación, **crearemos los objetos** ‘ventana’ y ‘tortuga’ (screen y turtle, respectivamente):

```
ventana = turtle.Screen()  
tortuga = turtle.Turtle()
```

Seguidamente, crearemos la **función ‘figura’** cuyo **parámetro será el número de lados** y cuya función será la de:

- “**Imprimir una línea de distancia 100,**
- y girar 360/lados ángulos” un número ‘lados’ de veces.

Finalmente, le **preguntaremos al usuario** el número de lados por pantalla y llamaremos a la función ‘figura’ con esos lados.

De esta forma, el código queda como sigue:

Código para pintar figuras geométricas

```
import turtle

ventana = turtle.Screen()
tortuga = turtle.Turtle()

def figura(lados):
    for i in range(lados):
        tortuga.forward(100)
        tortuga.left(360 / lados)

lados = input("Introduce el número de lados de la figura: ")

figura(int(lados))
ventana.exitonclick()
```

<https://www.w3schools.com/python/default.asp>

<https://www.programiz.com/python-programming/package>

SISTEMA DE GESTIÓN EMPRESARIAL

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Cómo funciona Odoo por dentro

Módulos

Modelos

Capa ORM

Vistas

Informes

Controladores

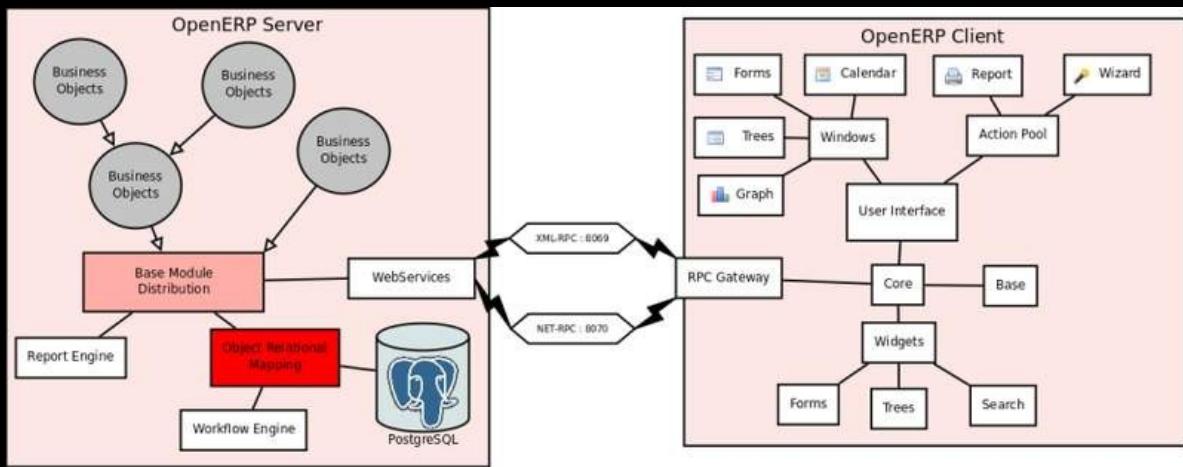
Campos calculados

Configuración de PyCharm para Odoo

Principios del funcionamiento de Odoo

Odoo usa una **arquitectura cliente/servidor** en la que los **clientes son navegadores web que acceden al servidor Odoo a través de RPC** (Remote Procedure Call o llamada a procedimiento remoto). El cliente se comunica con el servidor en **XML-RPC**, un protocolo de llamada a procedimiento remoto que usa **XML para codificar los datos** y **http** como protocolo de transmisión de mensajes.

Estructura cliente/servidor de Odoo



La **lógica y la extensión** del negocio, generalmente, se realizan en el lado del **servidor**.

Para comprender cómo funciona Odoo realmente por dentro, deberemos conocer lo siguiente:

- Existe una **capa ORM** (Object Relational Mapping) entre los objetos Python y la base de datos PostgreSQL, de forma que el **programador** no efectuará el diseño de la base de datos, este **creará clases** a las que la capa **ORM mapeará** con el SGBD. Gracias al ORM, no es necesario programar con **consultas SQL**, ya que esta capa proporciona una **serie de métodos** que lo facilita. Ahora, en lugar de hablar de tablas, se hablará de **modelos** que son **mapeados por el ORM en tablas**. Un modelo no es una tabla, es un **objeto con campos funcionales, restricciones** y campos **relacionales**.
- Odoo usa una **arquitectura modelo-vista-controlador (MVC)**:
 - **Modelo:** Clases diseñadas en **Python**.
 - **Vista:** Formularios, vistas, etc., definidos en **XML**.
 - **Controlador:** Reside en los **métodos** definidos en las **clases** que proporcionan la **lógica de negocio**.

Tanto las [extensiones](#) de servidor como las de cliente se empaquetan como **módulos** (módulo de compra, de ventas, CRM, facturación, etc.) que se **instalan y almacenan opcionalmente** en la base de datos.

Cada módulo de Odoo puede agregar una **lógica comercial nueva** o **alterar/ampliar** la lógica comercial existente en el sistema Odoo. “Todo en Odoo comienza y termina con módulos”.

Estructura interna de los módulos

Los módulos de Odoo amplían o modifican partes de modelo-vista-controlador.

De este modo, **un módulo puede tener**:

- Objetos de negocio:

Son la parte del **modelo**, están **definidos en clases** de Python según una sintaxis propia del ORM de Odoo.

- Vistas de objetos:

Definición de **visualización de IU** de objetos de negocio.

- Ficheros de datos:

Son **archivos XML o CSV** que pueden definir **datos, vistas, configuraciones**, etc.

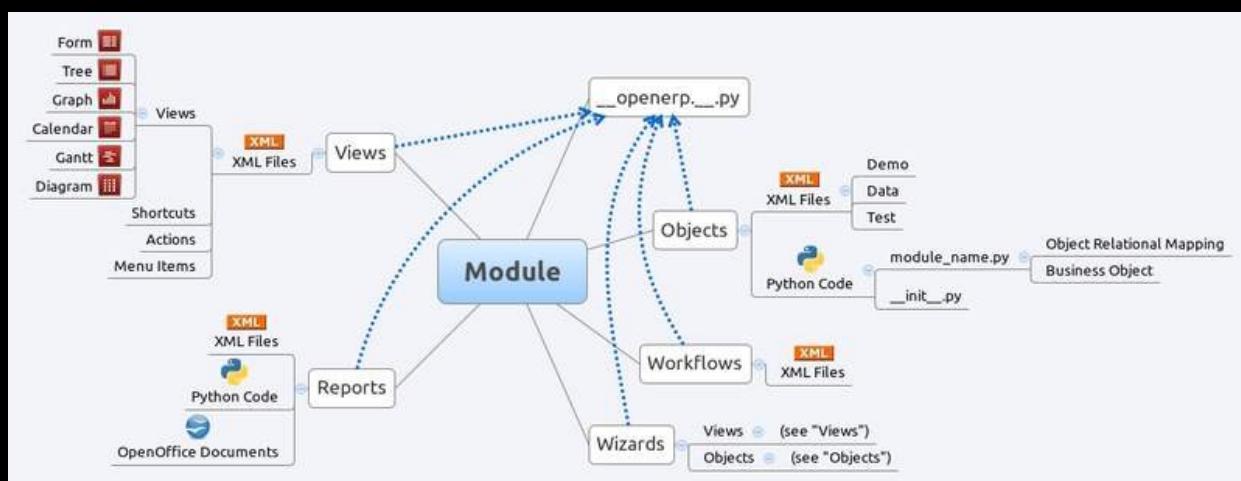
- Controladores web:

Gestionan las solicitudes de los navegadores web.

- Datos estáticos:

Como pueden ser **JavaScript, imágenes** u hojas de estilo en cascada (**CSS**) utilizados en la interface web.

Los módulos en Odoo



Los módulos requieren tener en su interior los siguientes archivos:

- ‘`__manifest__.py`’:

Sirve para declarar un paquete de Python **como un módulo** Odoo y para especificar **metadatos** del módulo. Además, ayuda a **mostrar** el módulo dentro de la **lista de aplicaciones** Odoo. Dentro de él, se especifica el **nombre** del módulo, la **versión**, una **descripción**, las **dependencias** con otros módulos, etc. Tienes más información aquí.

El archivo manifest

```
{  
    'name': "A Module",  
    'version': '1.0',  
    'depends': ['base'],  
    'author': "Author Name",  
    'category': 'Category',  
    'description': """"  
        Description text  
    """",  
    # data files always loaded at installation  
    'data': [  
        'views/mymodule_view.xml',  
    ],  
    # data files containing optionally loaded demonstration data  
    'demo': [  
        'demo/demo_data.xml',  
    ],  
}
```

- ‘`__init__.py`’:

En él, se implementan **instrucciones para importar archivos** Python.

Por ejemplo, si el módulo tuviese un solo archivo Python denominado “mimodulo.py”, el “init” tendría una sola línea de código ‘from . import mimodulo’.

Los módulos se encuentran en el directorio que se especifique en el ‘addons_path’ del archivo de configuración de Odoo (odoo.conf), pudiendo ser más de un directorio, como veremos este vídeo.

La estructura del módulo

En el presente vídeo, analizaremos de dónde se configuran los módulos con los que trabaja nuestro Odoo, y veremos un esqueleto básico de un módulo de Odoo. Como hemos comentado en la documentación, los módulos con los que va a trabajar nuestro Odoo se encuentran en el addon path del archivo de configuración de Odoo. Pueden ser varios, y en este caso, se están trayendo todos los módulos de esta ruta de addons, también de Odoo, mis módulos y de mis módulos, mi primer módulo.

Este es el archivo de configuración de Odoo, como ya sabemos, está dentro de esta ruta y un ejemplo de los módulos de un esqueleto de un módulo podría ser el siguiente. Este esqueleto lo hemos creado con el comando `scaffold` que más adelante comentaremos lo que es, pero adelantamos que crea un **esqueleto básico de un módulo** de Odoo. En este caso, le hemos puesto el nombre del módulo `ProductPrice2`, y como vemos, tiene nuestro archivo `_init_.py` que ahora mismo está vacío, y nuestro **manifiesto** que está informado, pero por defecto con un nombre, una descripción y demás. Esto lo tendremos que editar una vez que comencemos a crear este módulo.

Además, tenemos la carpeta `views` donde están nuestros módulos de vista que son los XML como hemos visto, los modelos, los controladores, etc.

Los modelos y la capa ORM (Object Relational Mapping)

Odoo mapea sus objetos en una base de datos con la capa ORM, evitando así tener que realizar consultas SQL. De esta forma, la **capa ORM** de Odoo facilita unos **métodos** que se encargan del **mapeo** entre los **objetos Python** y las **tablas de PostgreSQL**, por ejemplo, para crear, modificar, eliminar y buscar registros en la base de datos.

Los **modelos** son una manera de **relacionar** el **programa** con la **base de datos**. Realmente, para comprender qué son, podríamos asemejarlos a las tablas de la base de datos en Odoo, aunque no lo sean realmente. Se crean como **clases de Python** que pueden **extenderse** de:

- La clase `models.Model` para modelos **persistentes** en la base de datos.
- La clase `models.TransientModel` para datos **temporales** almacenados en la base de datos, pero que se borran automáticamente cada cierto tiempo.
- La clase `models.AbstractModel` para **superclases abstractas** destinadas a ser compartidas por múltiples modelos que la heredarán.

Los modelos **contendrán** los **campos y métodos** útiles para manejar el ORM. Al crearlos, es necesario dar **valores a algunas variables**, como `_name`, que es obligatoria y la que da nombre al modelo en Odoo. De esta forma, la definición mínima de un modelo sería la siguiente:

Definición mínima de un modelo

```
from odoo import models
class MinimalModel (models.Model):
    _name = 'test.model'
```

Los modelos son de **más alto nivel** que las consultas directas a **base de datos** y que las **clases y objetos** respecto a la **programación orientada a objetos**.

Unen en un único concepto las **estructuras** de datos, las **restricciones** de integridad y las opciones de **manipulación** de los datos.

Para ver los modelos existentes, se puede acceder a la base de datos PostgreSQL o mirar en **Configuración> Estructura de la base de datos> Modelos** dentro del modo desarrollador.

Fig. 7. Visualizando los modelos desde Odoo.

Los campos (fields) del modelo

Los campos (fields) o “columnas” se usan para indicar **qué puede almacenar** el modelo y dónde. Se implementan como **atributos** en la clase y se **declaran como un constructor**:

Declarando un campo en el modelo

```
from odoo import models, fields
class LessMinimalModel (models.Model):
    _name = 'test.model'

    name = fields.Char()
```

Pueden ser de **tipos**:

- De **datos simples**: Como integer, boolean, date, char, etc.
- **Reservados**: El **sistema gestiona estos campos** y no debe escribirse en ellos, solo leerlos si fuese necesario. Por ejemplo: id, create date, create_uid, etc.
- **Especiales**: Como **many2one**, **one2many**, **related**, etc.

Además, cuando se está creando un campo, se puede usar una serie de **atributos**, como **readonly**, **required**, etc.

Para **acceder a los campos**, debemos saber que las **interacciones con modelos y registros** se realizan a través de recordsets, que son una **colección ordenada** de registros del **mismo modelo**. La iteración en un recordset producirá nuevos conjuntos de un solo “**registro activo**” (**singletons o solteros**) del que se podrá **leer y escribir directamente**:

Recorriendo los registros

```
def do_operation(self):
    print(self) # ⇒ a.model(1, 2, 3, 4, 5)
    for record in self:
        print(record) # ⇒ a.model(1), then a.model(2), ...
```

También, se puede acceder a los valores de los campos **como elementos diccionario**. Establecer el **valor de un campo** desencadena una **actualización de la base de datos**:

Accediendo al campo por medio de elementos diccionario

```
>>> record.name = "Bob"
>>> field = "name"
>>> record[field]
```

Bob

Para ver los campos que tiene un modelo en Odoo, bastará con ir al modelo concreto y pulsar sobre él.

Campos de un modelo

Nombre de campo	Etiqueta de campo	Tipo de campo	Requerido	Sólo lectura	Indexado	Tipo	
activity_date_deadline	Siguiente plazo de actividad	fecha	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Campo base	
activity_exception_decoration	Decoración de Actividad de Excepción	Selección	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Campo base	
activity_exception_icon	Icono	Carácter	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Campo base	
activity_ids	Actividades	one2many	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Campo base	

Archivos de datos

Cuando creamos un módulo de Odoo, se pueden **definir datos** que se guardarán en la base de datos. Estos datos pueden ser **necesarios** tanto para:

- el **funcionamiento** del módulo,
- como de **demonstración**,
- o, incluso, parte de la **vista**.

Todos los **archivos de datos** están en **formato XML** con elementos <record>, siendo cada elemento <record> un **registro** de base de datos. Por ejemplo, en el módulo ‘Empleados’, habrá una **carpeta ‘data’** con un XML con todos los datos de todos los empleados. Los XML tienen una estructura básica como la siguiente:

Estructura básica de un archivo de datos

```
<odoo>

    <record model="{model name}" id="{record identifier}">
        <field name="{a field name}">{a value}</field>
    </record>

</odoo>
```

Donde:

- **model** es el nombre del **modelo** Odoo para el registro. Es **obligatorio** indicar uno.
- **id** es un **identificador externo**, permite hacer referencia al registro. **Se recomienda encarecidamente** informarlo.
- Los **fields** tienen un atributo ‘name’, y es el nombre del campo en el modelo (ejemplo: descripción). Su **cuerpo será el valor** de dicho campo. Es **obligatorio** informar la **etiqueta ‘name’**.

Los **archivos de datos** deben **declararse en el** archivo **manifest** del módulo.

Pueden declararse en:

- la lista de ‘**datos**’ (**siempre** cargada),
- o en la lista de ‘**demonstración**’ (solo cargados en modo de demostración).

Incluyendo los archivos de datos en el `__manifest__.py`

```
'data': [
    'archivo_de_datos.xml',
    'otro_archivo_de_datos.xml',
    'otro_archivo_mas_de_datos.csv'
]
```

Estos archivos XML son **flexibles y autodescriptivos**, y **muy detallados** cuando se crean varios registros simples del mismo modelo en masa.

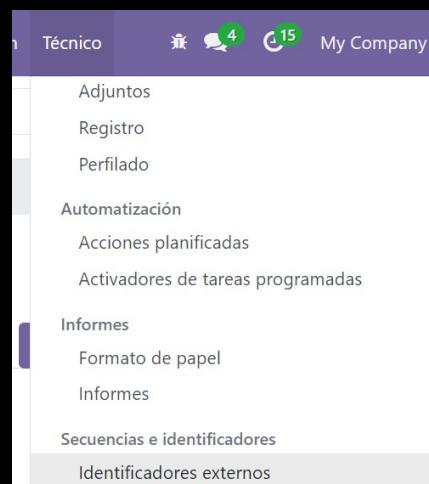
Para este caso, los archivos de datos **también pueden usar el formato CSV**, sabiendo que:

- el nombre del archivo será **model_name.csv**.
- la **primera fila** enumera los **campos** para escribir.
- cada fila crea **posteriormente** un nuevo **registro**.

Todos los '**record**' de la base de datos tienen un **identificador único** en la tabla, el **id**. Es un **contador automático** asignado por la base de datos. Sin embargo, si queremos hacer referencia a él en ficheros de datos u otros lugares, no siempre tenemos por qué conocerlo. La solución de Odoo son los **IDs externos**, que es una **tabla que relaciona cada ID de cada tabla con un nombre**. Se trata del modelo `ir.model.data`.

Para encontrarlos, hay que ir a:

Settings > Technical > Sequences & identifiers > External Identifiers



Ahí encontramos la **columna Complete ID**.

Nuevo Identificadores externos		1-80 / 10000+ < > 🔍		
	Id. completo	Nombre mostrado	Nombre del modelo	ID de registro
<input type="checkbox"/>	account.1_capital	301000 Capital	account.account	22
<input type="checkbox"/>	account.1_cash_diff_expense	642000 Pérdida por diferenc...	account.account	36
<input type="checkbox"/>	account.1_cash_diff_income	442000 Ganancia por difere...	account.account	26

Vistas, relaciones y herencias

Vistas básicas

Las vistas son la forma en la que se representan los registros de los modelos para ser **mostradas a los usuarios finales**. Se especifican en **XML**, lo que significa que son **flexibles**, permiten un alto nivel de **personalización** y pueden **editarse independientemente** de los modelos que representan.

Cada vista se materializa en una **visualización distinta**:

- vistas de árbol,
- formularios,
- kaban -que son una mezcla de xml, html y plantillas Qweb-,
- de búsqueda,
- calendario
- o gráficos).

En caso de que **no declaremos** las vistas, se pueden **referenciar por su tipo** y Odoo generará una **vista de lista o formulario estándar** para poder ver los registros de cada modelo.

Las vistas **se declaran como un registro del modelo ir.ui.view**, por ejemplo:

Estructura básica de una vista en Odoo

```
<record model="ir.ui.view" id="view_id">
    <field name="name">view.name</field>
    <field name="model">object_name</field>
    <field name="priority" eval="16"/>
    <field name="arch" type="xml">
        <!-- view content: <form>, <tree>, <graph>, ... -->
    </field>
</record>
```

El **tipo de vista** está **implícito** en el **campo 'arch'**, concretamente en la raíz.

Relaciones entre modelos

Un registro de un modelo puede estar **relacionado con un registro de otro modelo**. Las relaciones en Odoo se llaman:

- many2one: si es de ‘muchos a 1’.
- one2many: las ‘1 a muchos’.
- many2many: ‘muchos a muchos’.

Como sabemos, las relaciones ‘**muchos a muchos**’ en una base de datos relacional implican una tercera tabla intermedia, pero en Odoo no tenemos que preocuparnos de esto, ya que el **ORM simplifica las relaciones**. El mapeado de los objetos **detectará la relación y creará las tablas, claves y restricciones** de integridad necesarias.

Herencia

En el siguiente audio, hablaremos sobre las herencias de modelos en Odoo y cómo puede afectar a cada componente del MVC (modelo-vista-controlador):

Las herencias en Odoo

El framework de Odoo ofrece el mecanismo de la herencia para que los programadores puedan **adaptar módulos existentes** y garantizar a la vez que las **actualizaciones** de los módulos **no rompan los originales**.

La herencia se puede aplicar en los tres componentes del MVC:

- En el **modelo**: posibilita **ampliar las clases** existentes o **diseñar nuevas** clases a partir de las existentes.
- En la **vista**: posibilita **modificar el comportamiento** de vistas existentes o **diseñar nuevas** vistas.
- En el **controlador**: posibilita **sobrescribir los métodos** existentes o diseñar **otros nuevos**.

Odoo proporciona **tres mecanismos de herencia**:

1. la herencia de **clase**,
2. la herencia para **prototipo**,
3. y la herencia **por delegación**.

Informes y controladores

Los informes o reports

Los informes son los archivos que imprimiremos, almacenaremos o enviaremos por e-mail. El motor de informes utiliza una combinación de [Twitter Bootstrap](#), [QWeb](#) y [Wkhtmltopdf](#).

Consta de [dos partes](#):

- Un [ir.actions.report](#) para el que se suministra un **elemento de acceso <report>** y establece varios **parámetros** básicos para el informe:
 - como el **tipo predeterminado**,
 - si el informe debe **almacenarse en base de datos** tras la creación,
 - etc.
- Una vista Qweb para el contenido.

Ya que los informes son **páginas web estándar**, se puede acceder a ellos a través de una **URL**. Además, los **parámetros de salida** se pueden **modificar** a través de esta URL, por ejemplo, la **versión html** del informe de factura es accesible por medio de la siguiente URL:

http://localhost:8069/report/html/account.report_invoice/1

y la **versión en PDF** por medio de la siguiente:

http://localhost:8069/report/pdf/account.report_invoice/1

Informe de Odoo

1 / 1

97%

+



My Company (San Francisco)
Gran vía 14
28200 Madrid (Madrid)
España

Azure Interior
4557 De Silva St
Fremont CA 94538
Estados Unidos
Identificación fiscal: US12345677

Factura INV/2024/00001

Fecha de factura:
01/01/2024

Fecha de vencimiento:
29/02/2024

Fecha de envío:
01/01/2024

Descripción

[FURN_6741] Gran mesa de reuniones

Cant

5,00 4.000,00 15 %

Cantidad
\$ 20.000,00

[FURN_8220] Escritorio para cuatro personas
Estación de trabajo de oficina moderna para cuatro personas

5,00 2.350,00 15 %

\$ 11.750,00

Condiciones de pago: fin del siguiente mes

Importe neto

\$ 31.750,00

Comunicaciones de pago INV/2024/00001

Impuesto del 15 %

\$ 4,702.50

Vista

• 30.5 ± 3.0

Total

Los controladores

Todos los frameworks suelen tener controllers, y Odoo tiene los suyos. Los controladores de un módulo de Odoo son **clases** capaces de **capturar las solicitudes http** enviadas por cualquier navegador y que se usan cuando un **usuario quiere procesar algunos datos** en un sitio web. En Odoo, los controladores se utilizan para **configurar los módulos frontend**. Estos módulos frontend están **integrados con** módulos **backend**.

Por ejemplo, un usuario no puede usar la funcionalidad de “Models” en Odoo para representar los detalles del pedido de ventas en el sitio web. Sin embargo, **al usar el controlador**, los usuarios pueden **obtener** fácilmente los **detalles** del pedido de ventas **del backend**.

Los módulos como “[blog](#) del sitio web”, “[venta](#) del sitio web” y “[foro](#) del sitio web” están utilizando **controladores para ampliar sus funcionalidades**.

Los controladores [se crean heredando](#) del **Controller**.

Las [rutas](#) se definen a través de **métodos decorados con route()**:

Creando un controlador

```
from odoo import http

class MyController(odoo.http.Controller):
    @route('/some_url', auth='public')
    def handler(self):
        return self.stuff()

    def stuff(self):
        # Aquí deberías colocar la lógica de tu controlador
        # Puede ser cualquier cosa que deseas devolver como respuesta
        return "Hello from MyController!"
```

Se utiliza el decorador `@route` para definir la ruta ('/some_url') y la **autenticación** (`auth='public'`) para el método `handler`.

Con los controladores se puede realizar solicitudes http, solicitudes Json, enrutamientos, renderizados, etc.

Campos calculados

Es posible que, a la hora de mostrar los campos, **en lugar de traernos el valor de la base de datos**, el campo sea un campo calculado (**haciendo uso de otros campos**).

El valor del campo no se recupera de la BBDD, sino que se calcula sobre la marcha **invocando a un método del modelo**.

Para implementar este tipo de campos, bastará con **crear un campo** e informar su atributo 'compute' con el **nombre de un método**:

Ese método será el método de **cálculo**, que deberá establecer el **valor del campo** para calcular en cada registro en el atributo '**self**'.

Creando un campo calculado llamado name

```
import random
from odoo import models, fields, api

class ComputedModel(models.Model):
    _name = 'test.computed'

    name = fields.Char(compute = '_compute_name')

    def _compute_name(self):
        for record in self:
            record.name = str(random.randint(1, 1e6))
```

Explicación de este código:

`_name`: Especifica el nombre del modelo en Odoo.

`name`: Este es el campo calculado. Se define como un campo de tipo Char y utiliza el decorador `@api.depends()` para indicar que el método `_compute_name` es el encargado de calcular su valor.

`_compute_name`: Este método calcula el valor del campo `name`. En este caso, simplemente genera un número aleatorio entre 1 y 1 millón y lo convierte a cadena para asignarlo al campo `name` del registro actual.

Recuerda que cuando se accede al campo `name`, automáticamente se ejecuta el método `_compute_name` para calcular su valor. Si bien este ejemplo utiliza un valor aleatorio, en situaciones más prácticas, este método podría depender de otros campos en el mismo modelo o incluso en modelos relacionados, permitiendo cálculos más complejos y específicos según las necesidades del negocio.

Configurando PyCharm para trabajar con Odoo

Para los dos siguientes temas (14 y 15), vamos a necesitar tener configurado el entorno de desarrollo PyCharm (con el que ya trabajamos en temas anteriores) para trabajar con Odoo.

Para ello, **desde PyCharm**, abriremos la **carpeta** donde está **instalado Odoo** con:

File/Open

pero, ojo, no la carpeta que contiene el código fuente de Odoo (odoo.bin, odoo.conf, etc.), sino **C:\Program Files (x86)\Odoo 13.0\server**.

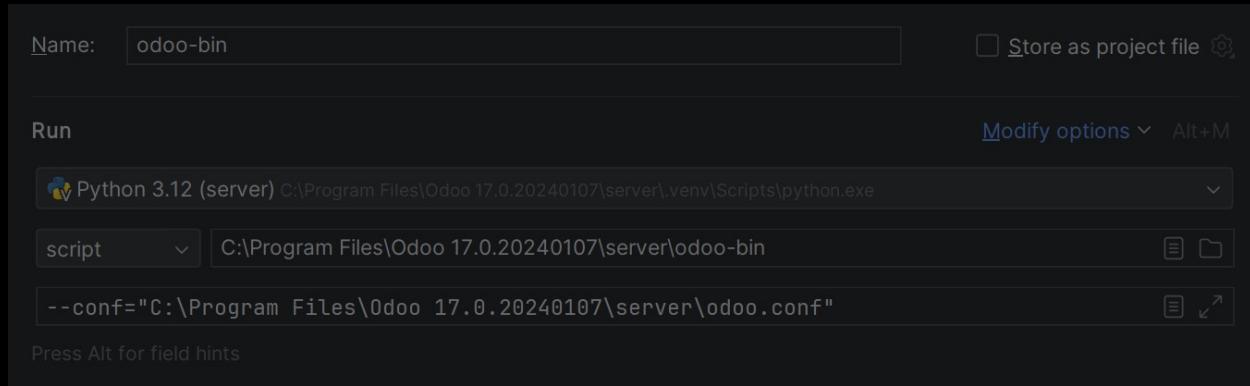
A continuación, crearemos un launcher para lanzar Odoo cuando necesitamos. Pare ello, en la parte superior derecha, pulsaremos sobre 'Add configuration'.

(En la versión 2023.3.3 no se ve esa pestaña, hay que buscar por: run/debug, y pulsar en la lista de resultados en "Edit Configuration ... run", y se abre la ventana, y pulsamos en el signo "+" de la izquierda superior. Ayuda: <https://www.youtube.com/watch?v=Uz8eguVc2LQ>).

Se nos abrirá una ventana en la que tendremos que informar:

- El nombre
- La ruta del script que queremos lanzar, que es el odoo.bin que tenemos en el rooth.
- El intérprete de Python; en nuestro caso, seleccionaremos la 3.7.
- Los parámetros, que informaremos con "--conf="C:\Program Files (x86)\Odoo 13.0\server\odoo.conf"" para que seleccione correctamente el fichero 'odoo.conf'. (Sustituir la ruta por la nuestra, al archivo odoo.conf, evidentemente).

Configurando el launcher

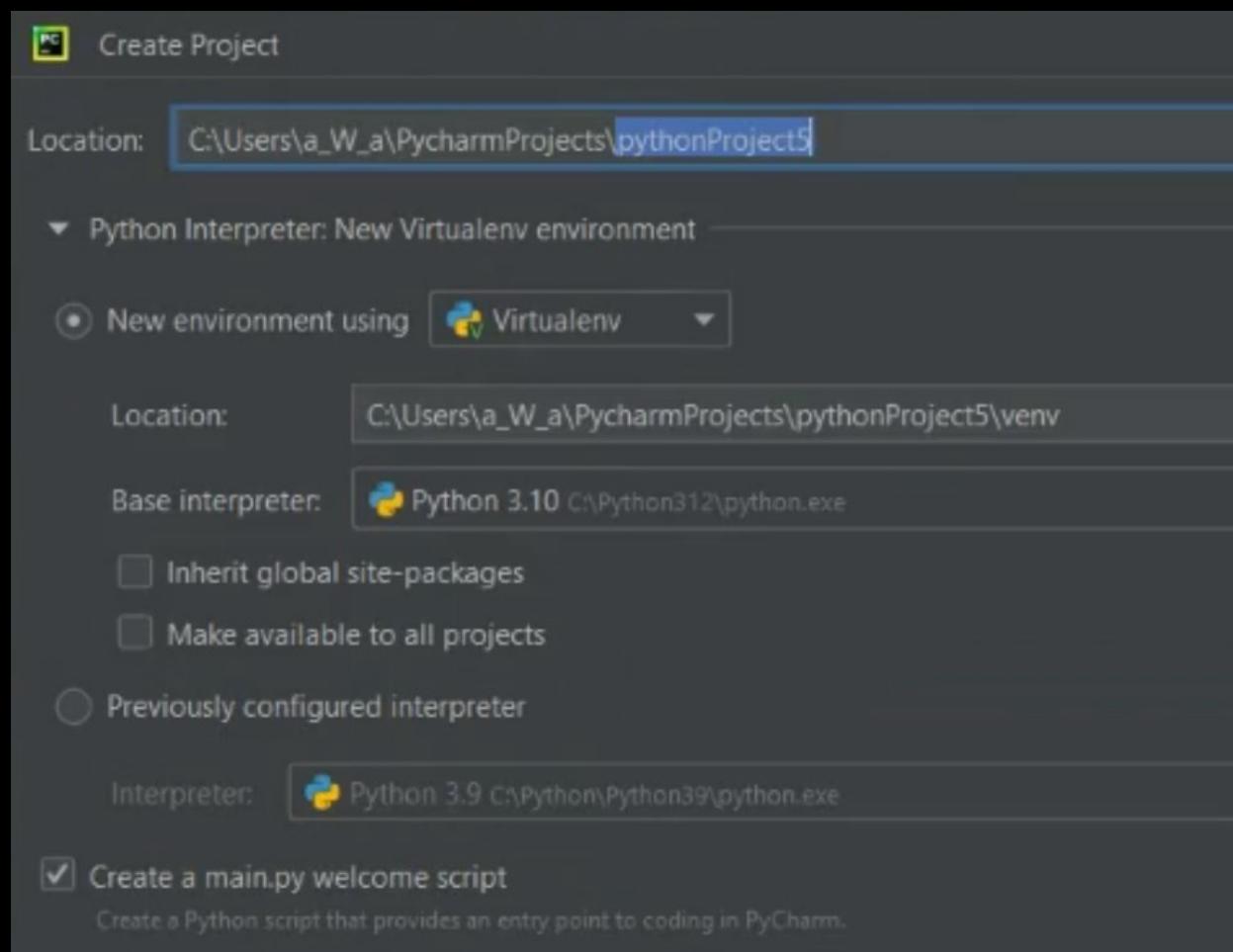


Al pulsar sobre el ícono play, comprobaremos que la conexión funciona correctamente según el log. No debemos alarmarnos de que aparezcan las letras en rojo.

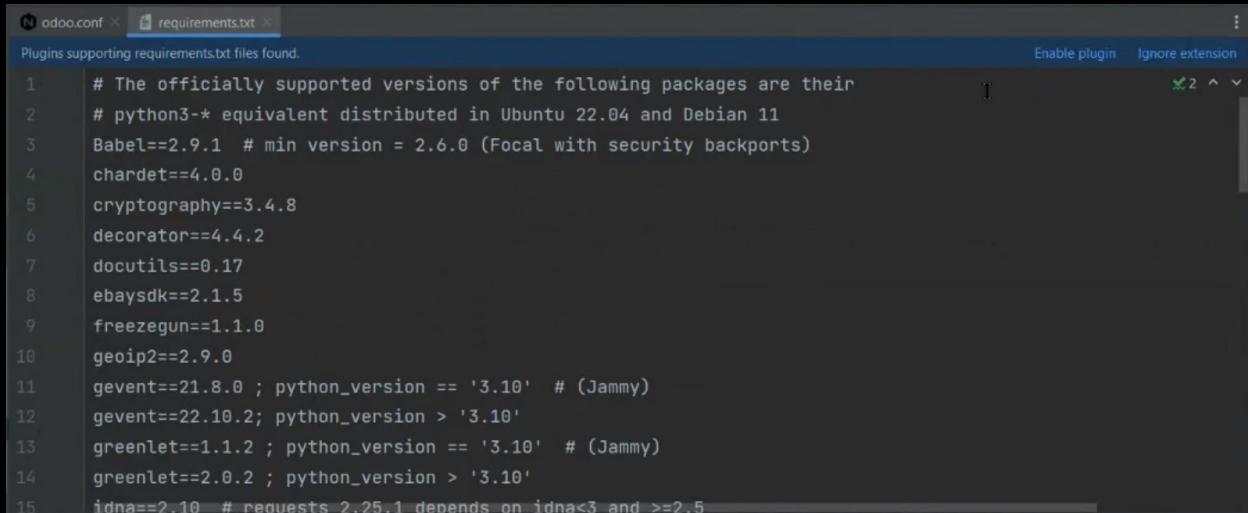
EXPLICACIÓN DEL PROFESOR: CONFIGURACIÓN DE PYCHARM CON ODOO

Vamos a abrir el PyCharm y les digo, si yo quiero enganchar el PyCharm con mi entorno de desarrollo, con el servidor, lo que tengo que hacer es:

1. Nuevo proyecto.
2. Al **nuevo proyecto**, decirle que vaya a la ruta donde tengo instalado Odoo, que en este caso es aquí, hasta **server**:
`C:\Program Files\Odoo 17.0.20231128\server`
Siguiendo las indicaciones, navego y llego hasta server.
3. Tengan cuidado con el **intérprete** que se pone, si no supera la versión 3.10 no le va a arrancar, entonces tienen que tener en cuenta eso.



Una vez que lo hacen, puede que le dé algún problema de que necesitan paquetes instalados, entonces hay un fichero que se llama `requirements.txt`, donde aquí tiene todos los requisitos que se necesita para poder arrancar. Le aparece: "instalar requisitos o requerimientos", y empiezan a descargarse todos estos paquetes. He tenido algún problema con este de aquí, que he tenido que cambiarle la versión (#Werkzeug==2.0.2 → Werkzeug==2.2.2).



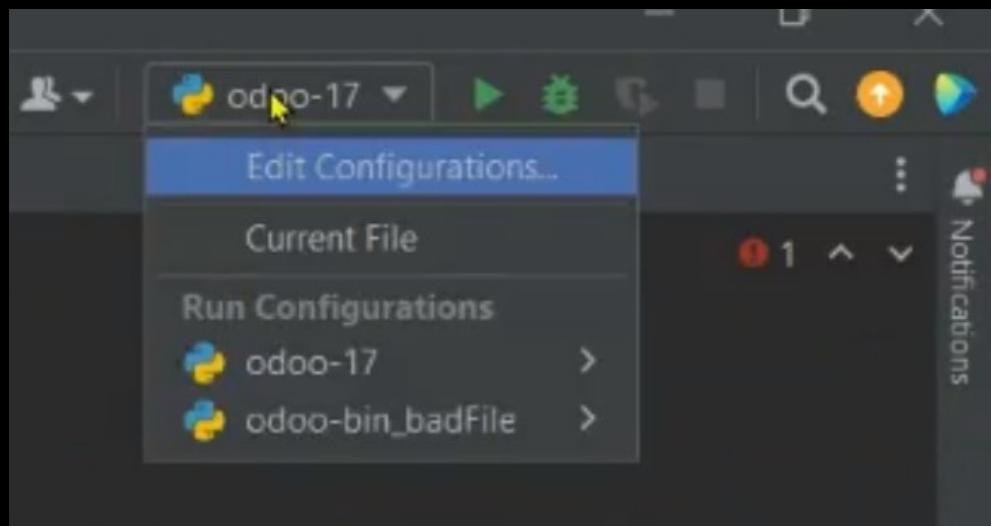
```
# The officially supported versions of the following packages are their
# python3-* equivalent distributed in Ubuntu 22.04 and Debian 11
Babel==2.9.1 # min version = 2.6.0 (Focal with security backports)
chardet==4.0.0
cryptography==3.4.8
decorator==4.4.2
docutils==0.17
ebaysdk==2.1.5
freezegun==1.1.0
geoip2==2.9.0
gevent==21.8.0 ; python_version == '3.10' # (Jammy)
gevent==22.10.2; python_version > '3.10'
greenlet==1.1.2 ; python_version == '3.10' # (Jammy)
greenlet==2.0.2 ; python_version > '3.10'
idna==2.10 # requests 2.25.1 depends on idna<3 and >=2.5
```

Y si ustedes necesitan **hacerlo a mano**, porque hay algún **paquete que no está aquí**, lo pueden añadir y volver a lanzar, sabiendo qué paquete es y con qué versión. Si no, se vienen aquí a Python Packages, y pueden buscar por ejemplo: py2d, lo que vayan queriendo y le dicen si está instalado o no está instalado, y si no lo tienen instalado, le dicen instalar paquete, y se va instalando para resolver todas las dependencias:



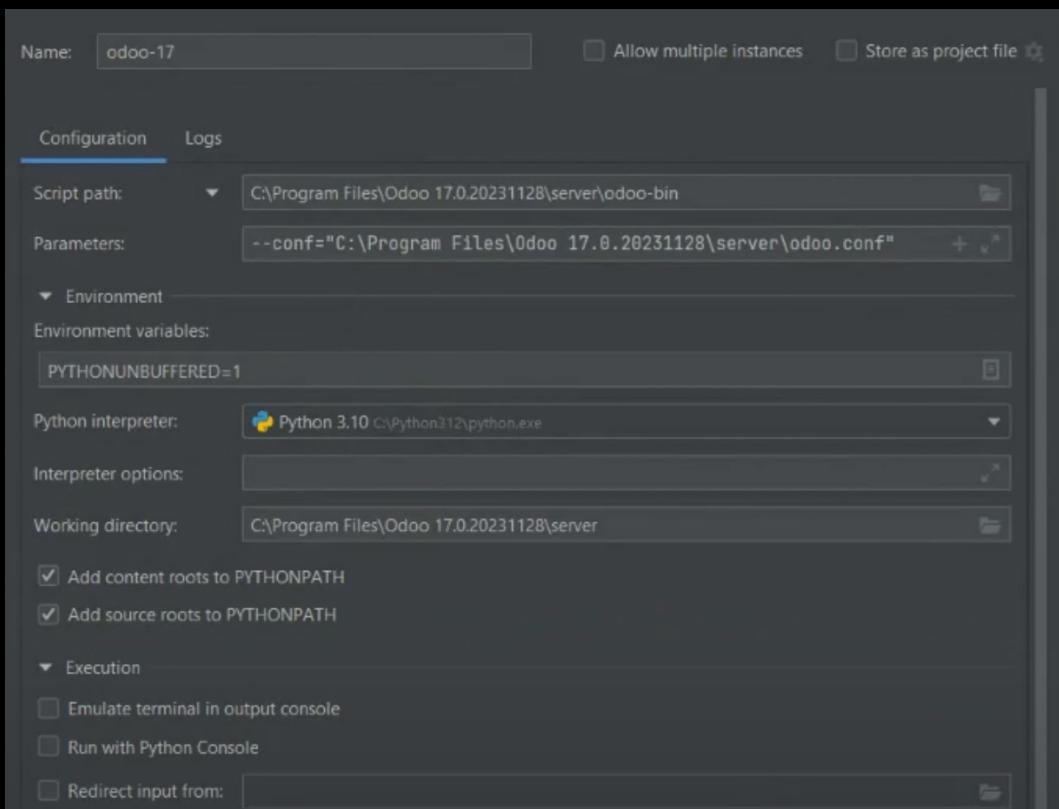
Una vez que está esto así, le dan a play.

Hay que configurar el lanzador, aquí:



Todos los datos que vienen puestos en el tema se aproximan y le sale:

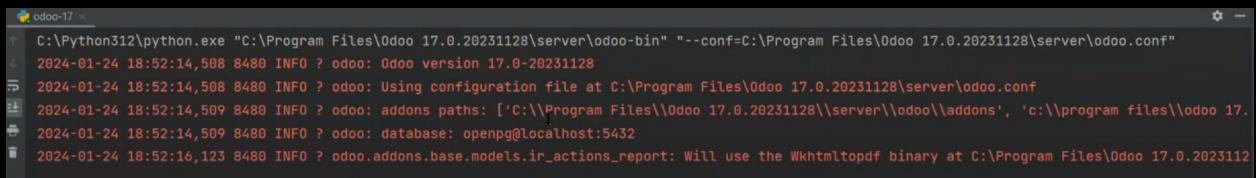
- el path, a donde lo tengan, de odoo-bin,
- le tienen que poner este parámetro, indicándole dónde está odoo.conf,
- tienen que poner este Environment, esta variable: PYTHONUNBUFFERED=1.
- El intérprete, insisto que la versión tiene que ser la apropiada.
- Una vez configurado, le dan a arrancar.



Tienen que ponerlo como archivo Python.

Entonces, cuando quiera arrancar, empezará a informar aquí de si está el servidor.

Qué está cargando... Qué está haciendo...



```
C:\Python312\python.exe "C:\Program Files\Odoo 17.0-20231128\server\odoo-bin" "--conf=C:\Program Files\Odoo 17.0-20231128\server\odoo.conf"
2024-01-24 18:52:14,508 8480 INFO ? odoo: Odoo version 17.0-20231128
2024-01-24 18:52:14,508 8480 INFO ? odoo: Using configuration file at C:\Program Files\Odoo 17.0-20231128\server\odoo.conf
2024-01-24 18:52:14,509 8480 INFO ? odoo: addons paths: ['C:\\\\Program Files\\\\Odoo 17.0-20231128\\\\server\\\\odoo\\\\addons', 'c:\\\\program files\\\\odoo 17.
2024-01-24 18:52:14,509 8480 INFO ? odoo: database: openpglocalhost:5432
2024-01-24 18:52:16,123 8480 INFO ? odoo.addons.base.models.ir_actions_report: Will use the Wkhtmltopdf binary at C:\Program Files\Odoo 17.0-20231128
```

Y cuando hagamos peticiones sobre el localhost, la petición que vamos a hacer al servidor, responde aquí con los mensajes.

Navegando por Odoo con PyCharm

Veremos cómo movernos por Odoo con la herramienta, con el software PyCharm. Después de haber configurado el proyecto con la carpeta de Odoo y haber configurado el launcher aquí, ya estamos en disposición de realizar modificaciones en PyCharm que se verán reflejadas en Odoo. Por ejemplo, aquí vemos el `odoo.conf`, que viene con la configuración de Odoo que le hemos indicado, el `odoo.bin` que será el ejecutable vinculado al launcher que será el que ejecutará nuestro Odoo. Si le damos al play, aquí veremos en un log que se conecta correctamente a nuestro Odoo. Para aprender un poco más a movernos por aquí, podemos ver en la carpeta de Odoo, aquí nos encontramos los módulos, en cuyo interior, por ejemplo, en el módulo `product` nos encontramos con la carpeta del modelo:

En primer lugar vemos que tiene tanto el manifest informado de su descripción, su nombre, su categoría de descripción, de dónde se traen los datos y demás, como su `__init__.py`, que en este caso tiene tres imports.

También nos encontramos aquí con los **modelos**, en cuyo caso, por ejemplo, en el modelo `Product Template` del módulo `Product`, podemos encontrarnos con la declaración de los campos, por ejemplo. Aquí es muy útil esta herramienta `Structure` (a la derecha) que te enseña la clase con todos sus métodos, los campos y demás.

Aquí podemos ver, por ejemplo, la **declaración de los campos en el Product Template** y además podemos buscar relación entre modelos, one to many, por ejemplo. Aquí vemos que este campo, `Packet It`, está relacionado con otro modelo, con un one to many.

También vemos **campos calculados** con la estructura `Compute`. Vemos que `Packet It` o `Packet`, este campo `Valid Product` al tener un **compute** es un campo calculado. Después de ver los modelos y los campos y las relaciones entre los modelos, campos calculados y demás, también podemos ver los datos. Aquí, como podemos observar, hay dos archivos `.xml` con los **datos record**.

Después también tenemos aquí las **vistas**. Todo esto dentro del módulo de `Product`. Aquí tenemos las vistas, las diferentes vistas de este módulo, los report, que también hemos hablado de ellos en este tema. En este caso no tiene carpeta de controles, pero, por ejemplo, el módulo `Project` sí tiene aquí su **carpeta de controles**. Con sus enrutamientos y demás.

Y con esto hemos hecho un repaso breve sobre lo que hemos tratado en este tema y visualizándolo desde PyCharm de Odoo.

Reflexión

Odoo tiene una serie de objetos interrelacionados entre sí, ajenos al usuario final, que hacen que disponga de esa interfaz gráfica tan sólida y amigable.

Cada módulo de Odoo tiene en su interior uno o varios modelos, que son archivos Python que relacionan el programa con la base de datos. Estos se ayudan de los métodos que ofrece la capa ORM para evitar tener que realizar consultas SQL. Además, deben tener vistas, que son archivos XML que representarán los registros de los modelos para ser mostrados a los usuarios finales. Esos registros están almacenados en los archivos de datos, que son documentos XML con elementos <record>, siendo cada uno de estos elementos un registro de base de datos. Además, existe la posibilidad de relacionar modelos, de implementar informes, campos calculados o crear controladores.

SISTEMA DE GESTIÓN EMPRESARIAL

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Creación de un módulo en Odoo

scaffold

Herencia

Log de Odoo

Errores en Odoo

Vamos a aprender a crear un módulo desde cero, en el cual, tendremos que implementar todos sus objetos (modelos, vistas, etc.), desde el inicio.

Veremos la **herencia de modelos y vistas ya existentes**.

Y hablaremos del **log de Odoo y los errores**, y cómo PyCharm nos ayudará a corregirlos.

Creación de un módulo desde cero

En el presente punto describiremos los pasos necesarios para crear un módulo desde cero, que consistirán en:

1. Creación del módulo

- Crear un repositorio para almacenar los nuevos módulos.
- Agregar los **nuevos módulos** al addon_path.
- Crear el **esqueleto/plantilla** del módulo (scaffold).

2. Explotación del módulo

- Configurar e implementar el nuevo módulo.
- Instalar el nuevo módulo en Odoo y verificar su correcto funcionamiento.

Crear un repositorio para los módulos:

Es recomendable el uso de un repositorio para almacenar todos nuestros **módulos personalizados**. A este repositorio le llamaremos, por ejemplo, “mismodulos” y lo crearemos dentro de:

“C:\Program Files (x86)\Odoo 13.0\server\odoo”.

Ahí será donde crearemos las **distintas carpetas para los distintos módulos**.

Agregar los nuevos módulos al addons_path:

Una vez creado el repositorio, **deberemos** agregar la ruta de éste en el addons_path del fichero de configuración “**odoo.conf**”, cada uno de ellos **separados por ‘,’ (coma)**. Haremos lo mismo con los futuros módulos que creemos en este repositorio.

El comando scaffold: Este comando sirve para crear un **esqueleto/plantilla de módulo** con la **estructura** básica de un módulo de Odoo, donde aparecerá **código de ejemplo** comentado como ayuda. Su **uso no es obligatorio**, pero **facilita** la creación de nuevo módulos.

Hay varias **formas de ejecutar el scaffold desde el terminal**, pero en nuestro caso haremos uso de la siguiente:

```
<ruta donde se encuentra Python.exe> +  
<ruta donde se encuentra odoo-bin> +  
scaffold <nombre que tendrá el módulo> +  
ruta donde se almacenará el módulo.
```

Las rutas entre comillas dobles y el nombre del módulo sin comillas.

Practicando con scaffold

Práctica con el comando Scaffold en Odoo desde PyCharm y Terminal

Practicaremos con el comando scaffold. Lo haremos tanto desde la aplicación PyCharm como desde el terminal.

Por ejemplo, esta es nuestra ruta de los módulos personalizados, que estarán en Odoo, mis módulos, y si por ejemplo quisiera crear un **nuevo módulo** con el comando scaffold, bastará con escribir en la terminal, como hemos visto:

- tanto la ruta donde se encuentra el ejecutable de Python,
- como la ruta donde se encuentra odoo-bin,
- seguido de scaffold,
- el nombre que queremos que tenga el nuevo módulo,
- y finalmente en qué lugar queremos que se almacene el nuevo módulo.

Por ejemplo, si ejecutamos, al pasar un momento vemos como si refrescamos la estructura de carpetas, vemos que hemos dicho que se guarde en server Odoo, mis módulos, el módulo que le hemos llamado librería.

También podemos hacerlo desde el terminal de Windows con el mismo comando, es decir, abrimos el terminal, escribimos el mismo comando, le vamos a cambiar, en este caso le vamos a poner el nombre librería2, y al ejecutarlo vemos como el comando scaffold actualiza, crea el nuevo módulo, librería2.

Estos como hemos visto tienen el **esqueleto básico que Odoo** nos proporciona para crear un nuevo módulo. Estos se encuentran también aquí, se han creado en el directorio concreto, con su carpeta de control, los controladores, los modelos, las vistas y el manifiesto que podemos abrir por ejemplo desde aquí mismo. Viene con una información por defecto que nosotros deberemos de modificar y al igual que los modelos vienen también con código de demostración comentado que nos puede servir de ayuda para ver lo que necesitamos escribir en estos objetos.



```
Microsoft Windows [Versión 10.0.18363.1016]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\Program Files (x86)\Odoo 13.0\server>"C:\Program Files (x86)\Odoo 13.0\python\python.exe" "C:\Program Files (x86)\Odoo 13.0\server\odoo-bin" scaffold libreria2
C:\Program Files (x86)\Odoo 13.0\server\odoo\mismodulos\libreria2
```

Configuración y verificación del módulo

Configurando e implementando el nuevo módulo:

Una vez creado el esqueleto del módulo, lo siguiente será **informar el archivo 'manifest.py'**, con campos como:

- 'name' con el nombre que queramos que tenga el **nuevo módulo** en la lista de aplicaciones.
- 'summary' con la **descripción corta** de lo que hará el módulo.
- 'description' con la descripción que queramos que aparezca en la **información del módulo**.
- 'author' y 'website' con el autor y la página web que queremos que aparezca en la información del módulo.
- 'category' con la categoría del módulo de entre las que podremos encontrar en la siguiente web:

https://github.com/odoo/odoo/blob/13.0/odoo/addons/base/data/ir_module_category_data.xml

- 'version' donde iremos **incrementando** la versión según vayamos realizando modificaciones.
- 'depends' con los **módulos** (no los modelos) que **necesitaría** como requisito nuestro nuevo módulo para funcionar, (y según un test y chatpt: pudiendo estar vacío, tener uno o varios).
- 'data' con los **datos y vistas** que se usarán en el módulo.
- 'demo' con los **datos** que se usarán cuando se esté ejecutando Odoo en modo **demostración**.

Ya solo quedaría **desarrollar los modelos, vistas, controladores**, etc. que sería la parte más compleja.

Verificando el funcionamiento del nuevo módulo en Odoo:

Para que nos aparezca el nuevo módulo en la lista de aplicaciones de Odoo, deberemos pulsar sobre '**actualizar lista de aplicaciones**'.

A continuación, buscaremos nuestro nuevo módulo, que lo hemos llamado 'librería'.

Desde ahí podremos instalarlo (lo ampliaremos más adelante). Si clicamos sobre '**Aprenda más**' el navegador abrirá una nueva pestaña con la **web que hayamos informado** en el manifest. Si por el contrario pulsamos sobre '**información del módulo**', se nos mostrará la información que hemos modificado en el **manifest**.

Creación de un módulo con herencias

Hay veces que queremos realizar modificaciones sobre aplicaciones ya existentes, ya sean estándares o no. Para ello, Odoo ofrece las herencias, que son una característica que nos permite realizar **modificaciones sobre objetos** sin necesidad de realizar las implementaciones sobre ellos. Es decir, al heredar se crea una especie de capa sobre el objeto original con las modificaciones que se quiera implementar sobre éste, pero **sin modificar el original** y con la posibilidad de **instalarlo y desinstalarlo cuando se requiera**. O lo que es lo mismo, en lugar de modificar un módulo existente, creamos un **nuevo módulo para realizar las modificaciones oportunas** (agregar o quitar funcionalidad) sobre el original. Tal y como vimos, la herencia puede afectar a cada componente del MVC (modelo-vista-controlador).

Según un test:

Con la herencia, en lugar de modificar un módulo existente, creamos un **nuevo módulo** para realizar las **modificaciones oportunas** (agregar o quitar funcionalidad) **sobre el original**.

Antes de crear nuevos campos programando con herencia, vamos a **buscar** con la ayuda de PyCharm un **modelo concreto entre todos los módulos** de Odoo, puesto que hay veces que queremos copiar o modificar un modelo concreto, pero no sabemos dónde se ubica. Esto nos será de mucha utilidad a la hora de crear herencias.

Ver la ubicación de un modelo

Vamos a ver un truco para buscar dónde se encuentra ubicado el modelo concreto que estamos tratando, en qué módulo, por ejemplo. En este caso queremos hacer alguna modificación o copiar cualquier cosa de esta ventana.

Sabemos que el modelo es SALEORDER. Entonces, para buscar dentro de qué módulo está, porque hay veces que es complicado encontrar el código, podemos hacer uso de PyCharm de la siguiente forma:

Sabemos que va a estar dentro de una carpeta de SALE, de ventas, pero hay muchas.

Entonces podemos pulsar **botón derecho sobre la carpeta de addons**, que son los módulos, y buscar en el **path**. A continuación **marcaremos los archivos Python**, porque lo que necesitamos es buscar el lugar **dónde** se está **definiendo el modelo**.

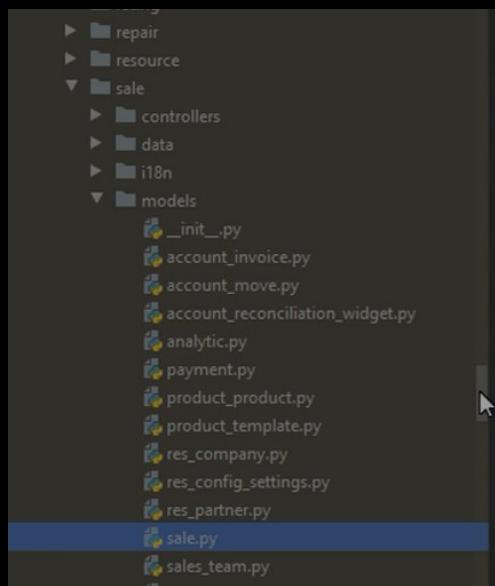
El modelo se define de la siguiente forma, con un

```
_name = SALEORDER
```

Si os fijáis solo encuentra una coincidencia que es esta, porque el modelo solo se define una única vez. Sin embargo, si en vez de 'SALEORDER' ponemos 'INERIT', aparecen muchas más, porque esto es **todas las herencias** de ese modelo.

Por ejemplo, si volvemos a la definición del modelo y pulsamos aquí doble clic, ya nos lleva al modelo en concreto donde se está declarando. Si pulsamos botón derecho File Path, nos muestra la ruta donde se encuentra concretamente el modelo. Es decir, está en la carpeta 'SERVER > OODO > ADDONS', en el módulo de 'SALE', la carpeta 'modelos' y 'SALE PATH'.

Es decir, la ruta sería la siguiente: 'SALE', en 'modelos', y aquí encontraríamos el modelo de 'SALE'. Que es el que, donde se está definiendo, el modelo que nos interesa.



Una vez que sepamos como encontrar el modelo del que vamos a heredar, empezaremos a trabajar la herencia con un ejemplo: Concretamente, en el **módulo de Ventas**, agregaremos un nuevo campo **debajo de 'Plazos de pago'** donde podremos informar el **'modo de pago'** de entre los indicados en un desplegable (metálico, transferencia bancaria o tarjeta).

En primer lugar, realizaremos los pasos que hemos visto en este tema hasta ahora, es decir,

- realizaremos un 'scaffold' para crear el **esqueleto del nuevo módulo**, al que llamaremos **'modos de pago'**,
- **borraremos** los objetos que en principio **no necesitaremos** (como, por ejemplo, las carpetas de 'controllers' y 'demo'),
- **quitaremos del archivo '__init.py'** las carpetas que **hayamos eliminado** previamente
- y, finalmente, informaremos el archivo **manifest**.

Ahora, ya estaremos preparados para **implementar el modelo** y la vista con herencias en lugar de implementarlos desde cero.

Herencia del modelo

Queremos agregar en el módulo de Ventas un nuevo campo debajo de ‘Plazos de pago’ donde podremos informar el ‘modo de pago’ de entre los indicados en un desplegable (métálico, transferencia bancaria o tarjeta).

Empezaremos creando el modelo, pero debemos hacerlo sin tocar el código estándar de Odoo, es decir haciendo uso de herencias.

Tal y como hemos aprendido, ya sabemos que tendremos que implementar una **herencia de ‘sales.order’**. Para ello, lo primero que haremos como **buenas prácticas** será **renombrar el modelo de la carpeta ‘model’** con el nombre del **modelo del que heredaremos** (en este caso `sale_order`), lo reemplazaremos en el ‘`__init__.py`’ de la **carpeta ‘models’**, y limpiaremos el código que se nos crea de plantilla del scaffold, **dejando** solo la línea con la importación de las **librerías básicas**.

Limpiando el código

The screenshot shows a code editor with a sidebar displaying a file tree. The tree includes 'mismodulos', 'libreria', 'miprimermodulo', 'modos_pago' (which contains 'models'), and two files: '_init_.py' and 'sale_order.py'. The '_init_.py' file is highlighted with a red box. The 'sale_order.py' file is also highlighted with a red box. The main editor area shows the following Python code:

```
# -*- coding: utf-8 -*-
from odoo import models, fields, api
```

A continuación, crearemos la **clase ‘SaleOrder’** como en la declaración del objeto original, solo que en este caso solo **informaremos** el **atributo ‘_inherit’**, para indicar que este modelo **es heredado** del modelo ‘`sale.order`’.

Finalmente, agregaremos el **campo ‘modo de pago’** de **tipo ‘selection’** con las tuplas:

- **Clave**: Lo que se guardará en base de datos.
- **Valor**: Lo que se mostrará en el desplegable.

Heredando el modelo y creándole el nuevo campo

```
from odoo import models, fields, api

class SaleOrder(models.Model):
    _inherit = 'sale.order'

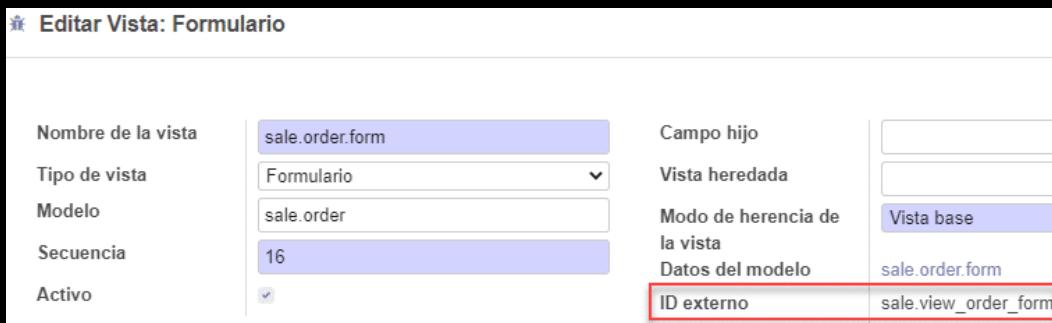
    modo_pago = fields.Selection(
        string="Modo de pago",
        selection=[
            ('efectivo', "Efectivo"),
            ('tarjeta', "Tarjeta"),
            ('transferencia', "Transferencia bancaria")
        ]
    )
```

Con esto ya tendríamos implementado el **modelo del módulo** con el **nuevo campo desplegable**, ahora nos quedará implementar la vista (para poder visualizar el campo) e instalar y probar el módulo.

Heredar la vista

En primer lugar, tendremos que buscar la vista de la que queremos heredar. Para ello, nos posicionaremos en Odoo en la ventana de Ventas **en la que queremos agregar el campo**, pulsaremos sobre la opción '**Editar Vista: Formulario**' del '*Open Developer Tool*' (el icono del insecto), y el campo '**ID externo**' nos indicará la vista de la que tendremos que heredar:

ID externo de la vista de la que queremos heredar



A continuación, en la carpeta '*views*' de nuestro módulo **eliminaremos** las vistas **ejemplo** que crea la plantilla y **crearemos una nueva** que se llamará '*sale_order_view.xml*'.

Seguidamente, agregamos esta **vista en el 'manifest'** en la sección '*data*', con el formato:

'data': ['views/sale_order_view.xml']:

Agregando la vista en el manifest

```
# any module necessary for this one to work correctly
'depends': ['sale'],

# always loaded
'data': [
    'views/sale_order_view.xml'
],
```

Ya solo nos quedará implementar el código de la vista en la que **heredaremos de la vista** **sale.view_order_form**.

En primer lugar, crearemos una **estructura <data>** para un archivo XML y en su interior haremos la **herencia** de la vista. En la herencia informaremos:

- el ‘name’ de la vista,
- el ‘model’ del que heredamos,
- y el ‘inherit_id’ con el ID externo (que vimos en odoo, en “editar vista formulario” del *Open Developer Tool*).

Finalmente, en el interior de la etiqueta ‘arch’ le indicaremos que el campo ‘**modo_pago**’ irá **después** del campo ‘**payment_term_id**’ (name=”payment_term_id”) y (position=”after”).

Heredando la vista y posicionando el nuevo campo

```
<?xml version="1.0" encoding="utf-8"?>
<odoo>
    <data>
        <!-- Inherit Form View to Modify it -->
        <record id="view_order_form" model="ir.ui.view">
            <field name="name">sale.order.form.modo.pago</field>
            <field name="model">sale.order</field>
            <field name="inherit_id" ref="sale.view_order_form"/>
            <field name="arch" type="xml">
                <field name="payment_term_id" position="after">
                    <field name="modo_pago"/>
                </field>
            </field>
        </record>
    </data>
</odoo>
```

Instalación del módulo

Antes de instalar el módulo, vamos a ver **cuántos campos** tiene el **módulo de Ventas**.

Para ello, en Odoo:

1. navegamos al módulo de **ajustes**,
2. abrimos **técnico**/modelos,
3. buscamos el **módulo 'sale.order'**
4. y vemos que dispone de 89 campos.

A continuación, instalaremos el nuestro y comprobaremos que ese número se habrá incrementado en 1 campo.

Para instalar el módulo, tal y como explicamos en el punto 3 del presente tema, bastará con ‘actualizar lista de aplicaciones’, buscar nuestro nuevo módulo ‘Modos de pago’ e **instalarlo**.

Tras su instalación, si volvemos a comprobar el número de campos que tiene el módulo de Ventas, ahora serán 90.

Entre los que se encuentra el nuestro ‘modo_pago’.

Si ahora abrimos el módulo de Ventas, comprobaremos que el campo se ha agregado correctamente en el lugar que queríamos.

La importancia de la herencia

Tal y como hemos visto, la herencia es algo muy importante para el programador de Odoo puesto que nos facilitará mucho el trabajo. Nos permitirá ir agregando o quitando piezas a un puzzle que puede ser todo lo grande o pequeño que queramos. Además, con la ventaja de que dichas piezas se pueden poner y quitar cuando sea necesario sin interferir el resto del puzzle.

La **funcionalidad base** del código estándar de Odoo **siempre estará ahí**, y podremos ampliarla haciendo uso de herencias.

El log de Odoo

Por defecto Odoo muestra todo el registro de log en el ‘stdout’ (**standar out** o salida estándar, que en PyCharm es el menú que aparece en la sección ‘run’), pero hay varias opciones para redirigir el registro a otros destinos y personalizar la **cantidad de salida de información** (todas **configurables** en el registro de **configuración odoo.conf**).

A continuación, comentaremos los más útiles de las descritas en la web de Odoo, en la sección logging:

logfile <archivo>:

Envía la salida de registro **al archivo especificado** en lugar de stdout. Por ejemplo, por defecto se informa en el directorio “C:\Program Files (x86)\Odoo 13.0\server\odoo.log”:

Si está **informado el parámetro logfile**, ese archivo **almacenará todos los logs** que se produzcan en Odoo en el nivel que **se indique en el log_handler**.

Si no está informado aparecerá en el stdout en PyCharm:

Stdout de PyCharm

```
C:\Users\msf\AppData\Local\Programs\Python\Python37\python.exe "C:/Program Files (x86)/Odoo 13.0/odoo-bin" --log-level=info  
2020-09-03 16:50:16,409 6496 INFO ? odoo: Odoo version 13.0-20200514  
2020-09-03 16:50:16,409 6496 INFO ? odoo: Using configuration file at C:\Program Files (x86)\Odoo 13.0\server\odoo.conf  
2020-09-03 16:50:16,409 6496 INFO ? odoo: addons paths: ['C:\\\\Program Files (x86)\\\\Odoo 13.0\\\\addons']  
2020-09-03 16:50:16,409 6496 INFO ? odoo: database: test@localhost:5432  
2020-09-03 16:50:16,912 6496 INFO ? odoo.addons.base.models.ir_actions_report: Will use the Werkzeug WSGI server  
2020-09-03 16:50:17,240 6496 INFO ? odoo.service.server: HTTP service (werkzeug) running on DESKTOP-1QHJL9A:8069
```

log_handler <:NIVEL>:

habilita el **log en el NIVEL proporcionado**, por ejemplo:

:DEBUG habilitará todos los mensajes de registro **en o por encima del nivel DEBUG**. Los niveles son:

- critical,
- error,
- warning,
- info,
- debug,
- notset.

Además de en el **archivo de configuración**, se puede tratar el manejador de error en cada ejecución **diferenciando objetos**:

```
$ odoo-bin --log-handler :DEBUG --log-handler odoo.models:CRITICAL --log-handler odoo.fields:WARNING
```

Analizando el log al ejecutar Odoo desde PyCharm

Al ejecutar Odoo desde PyCharm aparece una información en el log.

¿Qué quiere decir esta información? Por defecto el manejador se encuentra en :INFO ¿mostrará la misma información si lo ponemos en :DEBUG?

Así es, al ejecutar el intérprete de Odoo en PyCharm:

Ejecutando el intérprete



Nos aparece el siguiente log:

Log en modo :INFO

```
2020-09-03 16:50:16,409 6496 INFO ? odoo: Odoo version 13.0-20200514
2020-09-03 16:50:16,409 6496 INFO ? odoo: Using configuration file at C:\Program Files (x86)\Odoo 13.0\se
2020-09-03 16:50:16,409 6496 INFO ? odoo: addons paths: ['C:\\Program Files (x86)\\Odoo 13.0\\server\\odoo
2020-09-03 16:50:16,409 6496 INFO ? odoo: database: test@localhost:5432
2020-09-03 16:50:16,912 6496 INFO ? odoo.addons.base.models.ir_actions_report: Will use the Wkhtmltopdf I
2020-09-03 16:50:17,240 6496 INFO ? odoo.service.server: HTTP service (werkzeug) running on DESKTOP-4HNEI
```

Con la siguiente información:

- La **versión de Odoo**.
- La **ruta del archivo de configuración** (odoo.conf) que se está usando.
- La/s ruta/s de los **addons paths**, que están informadas en el archivo de configuración previo.
- La **base de datos** que se está usando.
- Se indica que se usará el **programa Wkhtmltopdf**.
- Se indica que donde está corriendo el **servicio HTTP**.

Si cambiamos el manejador a: DEBUG y volvemos a arrancar el intérprete, aparece más información:

Log en modo :DEBUG

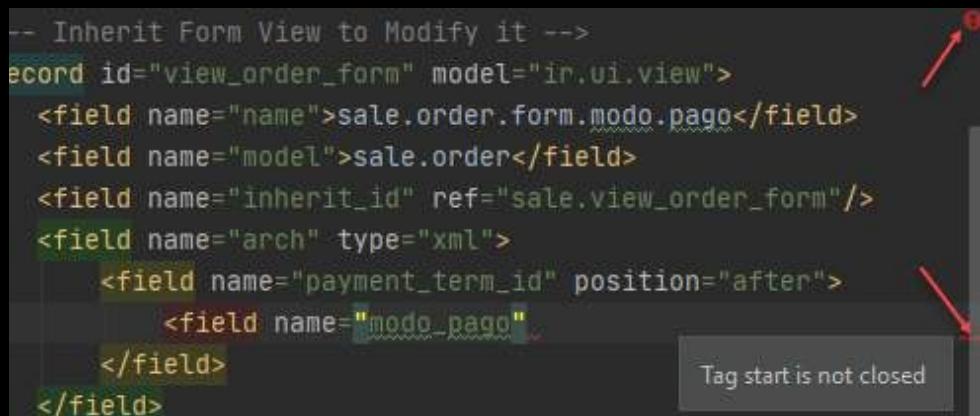
```
2020-09-03 17:30:28,094 8056 DEBUG ? odoo.netsvc: logger level set: "odoo.http.rpc.request:INFO"
2020-09-03 17:30:28,094 8056 DEBUG ? odoo.netsvc: logger level set: "odoo.http.rpc.response:INFO"
2020-09-03 17:30:28,094 8056 DEBUG ? odoo.netsvc: logger level set: ":INFO"
2020-09-03 17:30:28,094 8056 DEBUG ? odoo.netsvc: logger level set: ":DEBUG"
2020-09-03 17:30:28,094 8056 INFO ? odoo: Odoo version 13.0-20200514
2020-09-03 17:30:28,094 8056 INFO ? odoo: Using configuration file at C:\Program Files (x86)\Odoo 13.0
2020-09-03 17:30:28,094 8056 INFO ? odoo: addons paths: ['C:\\Program Files (x86)\\Odoo 13.0\\server\\'
2020-09-03 17:30:28,094 8056 INFO ? odoo: database: test@localhost:5432
2020-09-03 17:30:28,717 8056 INFO ? odoo.addons.base.models.ir_actions_report: Will use the Wkhtmltopdf
2020-09-03 17:30:29,060 8056 DEBUG ? odoo.service.server: Setting signal handlers
2020-09-03 17:30:29,060 8056 DEBUG ? odoo.service.server: cron0 started!
2020-09-03 17:30:29,060 8056 DEBUG ? odoo.service.server: cron1 started!
2020-09-03 17:30:29,075 8056 INFO ? odoo.service.server: HTTP service (werkzeug) running on DESKTOP-4H
2020-09-03 17:31:29,075 8056 DEBUG ? odoo.service.server: cron0 polling for jobs
2020-09-03 17:31:30,079 8056 DEBUG ? odoo.service.server: cron1 polling for jobs
```

Errores o Dumps

Puede darse el caso de que al programar cometamos **errores sintácticos**. En esos casos, PyCharm resaltará dichos errores de código en tiempo real, es decir, los mostrará a medida que se está escribiendo. Por ejemplo, si se nos olvidase cerrar una etiqueta en un archivo XML que estemos editando, PyCharm nos mostrará con:

- una **exclamación roja en la parte superior derecha** que ese objeto tiene errores
- y además nos indicará con una **línea roja** también, **dónde y cuál es el error** concreto:

Errores de código en tiempo real



The screenshot shows a portion of an XML file in PyCharm. The code is as follows:

```
-- Inherit Form View to Modify it -->
<record id="view_order_form" model="ir.ui.view">
    <field name="name">sale.order.form.modo_pago</field>
    <field name="model">sale.order</field>
    <field name="inherit_id" ref="sale.view_order_form"/>
    <field name="arch" type="xml">
        <field name="payment_term_id" position="after">
            <field name="modo_pago" />
        </field>
    </field>
</record>
```

A red exclamation mark is positioned above the opening tag of the innermost field element. A red underline is placed under the attribute value "modo_pago" of the inner field element. A tooltip window is open at the bottom right, displaying the message "Tag start is not closed".

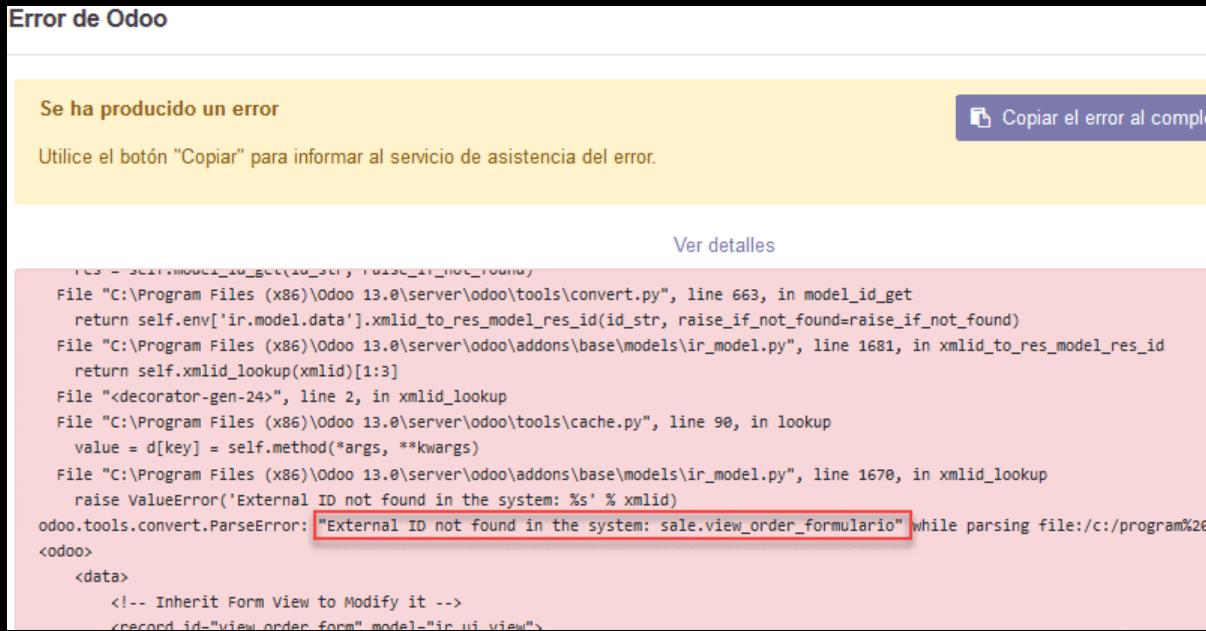
Una vez solucionado el error, la exclamación roja de la parte superior derecha pasará a ser un **tick verde**, que nos indicará que el **objeto actual no tiene errores** de sintaxis.

Código sin errores de sintaxis

```
-- Inherit Form View to Modify it -->
<record id="view_order_form" model="ir.ui.view">
    <field name="name">sale.order.form.modo_pago</field>
    <field name="model">sale.order</field>
    <field name="inherit_id" ref="sale.view_order_form"/>
    <field name="arch" type="xml">
        <field name="payment_term_id" position="after">
            <field name="modo_pago"/>
        </field>
    </field>
```

También pueden darse otros tipos de errores, como los **ortográficos**, que **no son capturados en tiempo real por PyCharm**. Por ejemplo, si en lugar de heredar en nuestro ejemplo anterior del ID externo de la vista padre 'sale. view_order_form' lo hiciésemos por error de 'sale.view_order_formulario' (que no existe), al intentar instalar el módulo será cuando Odoo nos muestre el error:

Error en tiempo de ejecución



Al analizar este error en tiempo de ejecución, podremos observar que nos está indicando que **no se ha encontrado el ID externo**. Esto nos indicará lo que deberemos solucionar para poder instalar el módulo de manera satisfactoria.

Hemos:

- Aprendido a crear un **módulo desde cero**.
- Creado un **repositorio** para los módulos personalizados y lo hemos **referenciado en el addon_path**.
- Conocido la funcionalidad y cómo usar el comando **scaffold**.
- **Configurado, implementado, instalado y probado** el nuevo **módulo**.
- Hecho uso de las **herencias**.
- Configurado y analizado algunos **logs y errores** que ofrece Odoo.

Podemos ofrecer la posibilidad a nuestro cliente de crearle un nuevo módulo que el podrá instalar/desinstalar cuando desee. Este nuevo módulo será un **módulo heredado** del módulo estándar de Odoo y le agregará la **nueva funcionalidad** que el cliente requiere.

De esta forma, nosotros simplemente crearemos un **nuevo módulo** en el cual implementaremos la herencia **desde el módulo estándar** y agregaremos la **nueva funcionalidad**. Así, cuando el cliente actualice la lista de aplicaciones, le aparecerá el nuevo módulo, con el cual **podrá instalar y desinstalar** la funcionalidad cuando quiera.

https://www.odoo.com/es_ES/

<https://praxyformaplus.com/>

<https://odooerpcloud.com/>

SISTEMA DE GESTIÓN EMPRESARIAL

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Administración e informes en Odoo

Usuarios y permisos

Qweb

Errores comunes

Agregar campos en formularios

En este punto agregaremos un **campo a un formulario** existente desde Odoo. Es simple, pero el problema es que, si actualizamos el módulo o actualizamos la versión de Odoo, este cambio se perderá.

Por ejemplo, al formulario de '**contactos**' le agregaremos el **campo "Facebook"**, justo **debajo del campo "enlace a página web"**.

Para ello, lo primero que deberemos hacer es colocar Odoo en **modo desarrollador**. A continuación, identificaremos qué modelo usa el formulario pulsando sobre 'metadatos'.

En nuestro caso es el 'res.partner'. Seguidamente, nos vamos al modelo en concreto (**ajustes/técnico/estructura de la base de datos/modelos**) y, tras **pulsar editar**, aparecerá al final de todos los campos un vínculo denominado '**agregar línea**', el cual pulsaremos. Esto nos abrirá una nueva ventana donde deberemos informar los **atributos del nuevo campo**.

Si observamos detenidamente, todos los **campos personalizados** que se crean desde el front-end **comenzarán su nombre con 'x_'**. Una vez **guardados** todos los cambios Odoo habrá **creado el nuevo campo en la base de datos**.

Notas propias:

En Odoo, puedes **ver qué modelo** está asociado a un formulario siguiendo estos pasos:

1. Inicia sesión en Odoo como administrador o usuario con permisos suficientes.
2. Navega a la **aplicación que contiene el formulario** del que deseas conocer el modelo asociado.
3. **Abre el formulario** en cuestión.
4. Una vez que estés en el formulario, puedes hacer **clic en el botón "Editar" o "Modificar"** si tienes permisos para editar los registros.
5. En la **URL del navegador**, verás algo como:

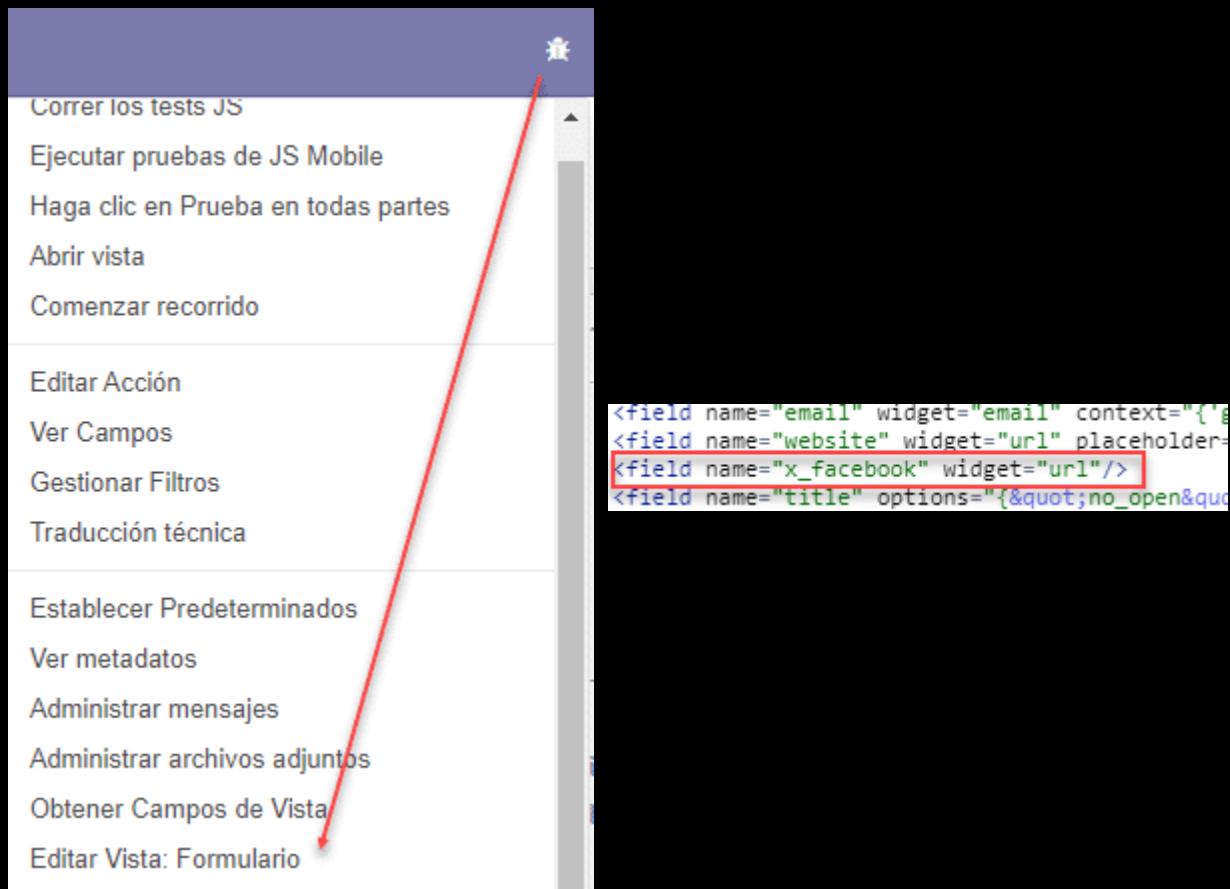
/web#id=123&view_type=form&model=sale.order, donde *sale.order* es el **modelo asociado** al formulario en este ejemplo.

6. Alternativamente, si tienes acceso al **modo de desarrollador** en Odoo, puedes habilitarlo y luego, al hacer **clic derecho** sobre cualquier **campo del formulario**, seleccionar la opción "**Inspectar**" o "Inspect" para ver el **código HTML asociado**. Dentro del código HTML, generalmente encontrarás la **referencia al modelo** asociado.

Agregar el nuevo campo a la vista del formulario

Para agregar el nuevo campo **a la vista** del formulario, volveremos al formulario de ‘contactos’ y nos posicionaremos sobre el campo “enlace a página web” para ver cuál es el **nombre del campo en concreto**, ya que el nuevo campo lo posicionaremos tras él. En este caso es ‘website’:

Una vez encontrado, pulsaremos sobre ‘**Editar Vista: Formulario**’ y buscaremos la etiqueta de ‘website’ para colocar justo debajo de la etiqueta del nuevo campo, tal y como se muestran en las figuras 7 y 8.



Como se puede observar, se le ha agregado el atributo **widget="url"** , ya que lo que contendrá, será un **vínculo a una página web**, con la funcionalidad de pulsar y navegar desde ese punto. Con esto ya habremos agregado a la vista el nuevo campo.

Finalmente, se guardarán todos los cambios y se refrescará el formulario. En ese momento, tal y como se aprecia en la figura 9, el nuevo campo ‘Facebook’ ya aparecerá y le podremos agregar la información.

Resultado de agregar el nuevo campo

The screenshot shows the Odoo partner form. At the top, there is a purple header bar with the word 'Facebook'. Below it, a black sidebar displays the configuration for the 'x_facebook' field: 'Field: x_facebook', 'Objeto: res.partner', 'Tipo: text', and 'Widget: URL (url)'. To the right of the sidebar, there is a contact card with fields for phone (+1 (650) 691-3277), email (info@yourcompany.example.com), and website (http://www.example.com). Below the contact card, there is a table with two rows: 'Facebook' and 'Categorías'. The 'Facebook' row contains the URL 'https://www.facebook.com/yourcompany', which is highlighted with a red box.

Verificación de la existencia del nuevo campo en la base de datos

Ahora podremos pasar a comprobar que el campo existe en la base de datos y que, además, el dato se ha almacenado correctamente.

Para ello, abriremos **PgAdmin** y desplegaremos la tabla, que en este caso es `res_partner`.

Tal y como se aprecia en la figura, el campo `x_facebook` **se ha agregado correctamente al final de la tabla** `res_partner`.

The screenshot shows the PgAdmin interface displaying the structure of the 'res_partner' table. The table has several columns: 'picking_warn_msg', 'buyer_id', 'purchase_warn', 'purchase_warn_msg', 'x_facebook', and two additional sections labeled '▶▶ Constraints' and '▶ Indexes'. The 'x_facebook' column is highlighted with a light blue background.

Ahora, para comprobar que se ha informado correctamente el valor, vamos a buscar **cuál es el ID** del elemento que hemos modificado para así **seleccionar solo ese registro** con una **sentencia** que selecciona a la tabla `res_partner`.

Tal y como ya sabemos, el ID lo podemos ver en los **metadatos** del elemento:

Buscando el ID del elemento en su metadata

The screenshot shows the Odoo metadata view for the 'res.partner' object. It displays the following fields: 'ID' (with value '1'), 'XML ID' (with value 'base.main_partner'), and 'Sin actualización' (with value 'True (cambiar)'). The 'ID' field is highlighted with a red box.

Ya solo nos queda realizar la **sentencia SQL** con la herramienta ‘**Query Tool**’ de PgAdmin para obtener la información.

The screenshot shows the PgAdmin Query Tool interface. The title bar reads "test/test@PostgreSQL 9.5 (x86)". Below it, there are two tabs: "Query Editor" (which is selected) and "Query History". The main area contains the following SQL code:

```
1  SELECT x_facebook
2  FROM public.res_partner
3  WHERE id = '1'
```

Below the code, there are four tabs: "Data Output" (selected), "Explain", "Messages", and "Notifications". The "Data Output" tab displays the results of the query in a table:

	x_facebook
text	
1	https://www.facebook.com/yourcompany

Exportar datos

Se necesita **extraer en un fichero Excel**, el nombre del producto, el coste, el precio de venta y la cantidad a mano (en ese orden).

Para ello, en el módulo de Ventas, nos dirigiremos a los productos. A continuación, abriremos la vista de lista (**View list**), **seleccionaremos todos los productos** y pulsaremos sobre **acción/exportar**:

The screenshot shows the Odoo Product List View. Step 1 highlights the 'Productos' tab. Step 2 highlights the search bar. Step 3 highlights the checkbox for selecting multiple products (checkboxes are selected for FURN_6666, Deposit, and FURN_0002). Step 4 highlights the 'Exportar' button in the toolbar.

	Nombre	Coste	Cantidad a mano	Cantidad pronosticada
1	FURN_6666	210,00	0,000	0,000
2	Deposit	100,00		
3	FURN_0002	0,00	0,000	0,000
4	EVO_H1	100,00		

Seguidamente, en la ventana emergente, **seleccionaremos los campos** que queremos exportar y su **orden**, el **formato** de salida del documento (XLSX en este caso) y finalmente pulsaremos sobre el botón 'Exportar'.

The screenshot shows the 'Exportar información' dialog box. Step 1 highlights the 'Formato de exportación' radio button for 'XLSX'. Step 2 highlights the 'Campos a exportar' section, which includes a list of fields: Nombre, Coste, Precio de venta, and Cantidad a mano, all of which are enclosed in a red box.

Dando como resultado un fichero Excel.

Administrador de usuarios

En el presente punto trataremos la administración de usuarios en Odoo, abordaremos un primer bloque exponiendo cómo crearlos y desactivarlos, y a continuación nos adentraremos en la gestión de permisos:

Crear usuarios: La creación de un nuevo usuario se llevará a cabo desde el **módulo de ajustes**, en '**usuarios y compañías/Usuarios**' o en '**administrar usuarios**':

Al pulsar sobre el botón 'Crear' lo primero que deberemos informar será el **nombre del nuevo usuario**, su dirección de **email** (el que usará para **iniciar sesión** en Odoo) y su **fotografía**.

Eliminar usuarios: Para eliminar un usuario bastará con seleccionarlo y pulsar en **acción/suprimir** (*Nota: no eliminar al usuario principal (admin)):

Gestión de grupos: Los Grupos en Odoo son bloques donde encapsular una **serie de reglas y permisos** para luego ser **asignados a usuarios**. Un **usuario puede tener asignado varios** grupos al igual que **un grupo puede estar asignado a varios** usuarios.

La creación, eliminación o edición de éstos se llevará a cabo desde el **módulo de ajustes**, concretamente en la sección '**usuarios y compañías/Grupos**'.

Odoo ya instala una serie de **grupos predefinidos** al instalar los módulos. De esta forma, por ejemplo, al instalar el módulo de ventas, se creará el grupo 'Ventas / administrador', el cual tiene en su interior una serie de reglas y permisos que permitirán dotar al usuario que tenga este grupo asignado una serie de permisos de administración en dicho módulo.

Gestión de permisos de los usuarios

Los permisos podremos gestionarlos tanto en el **momento de crear** el usuario, como **más adelante**.

Para ello, en primer lugar, deberemos entrar en el usuario en concreto al que queramos gestionarle los permisos (desde el módulo de ajustes, en 'usuarios y compañías'/Usuarios o en 'administrar usuarios'). A continuación, en la sección '**Permisos de acceso**' tendremos las siguientes **secciones**:

Multicompañía:

Porque un usuario puede estar o no en **diferentes compañías**.

Seleccionaremos las compañías permitidas y la empresa predeterminada.

Tipo de Usuario: En los que diferenciaremos al:

- '**usuario interno**' (aquel que trabajará internamente con Odoo),

- ‘Portal’
- y ‘Publico’.

A estos **dos últimos** no se **nos permitirá gestionar permisos**, ya tienen unas reglas preestablecidas.

<Módulos>:

Aparecerán los módulos que tengamos **instalados** en Odoo y los diferentes **niveles de permisos** a configurar dentro de ellos. Así, por ejemplo, si tuviésemos el módulo de Fabricación instalado, se podría configurar los **privilegios** en dicho módulo como:

- ‘administrador’,
- ‘usuario’
- o <blanco> (si no queremos que el usuario tenga acceso a este módulo).

Configuración técnica, permisos extra y otros (technical):

Finalmente se muestran una serie de opciones técnicas que podremos marcar/desmarcar para ofrecerle al usuario más o menos privilegios.

Al **guardar**, ya se nos habrá **informado los ‘Grupos’** con los **permisos** en el usuario, los cuales podremos editar.

Creación de usuarios y gestión de permisos

En este ejemplo, veremos cómo crear un usuario y gestionar sus permisos. Siguiendo los pasos indicados en los apuntes, nos dirigimos al módulo de **ajustes**, y seleccionamos la **sección de usuarios** para administrarlos. También podemos acceder desde "Usuario y Compañía" -> "Usuarios". Una vez dentro, visualizamos los usuarios existentes. Actualmente, hay dos usuarios internos y uno de portal creados.

Para crear uno nuevo, hacemos clic en "Crear". Como ejemplo, asignaremos el nombre "test2" y una dirección de correo electrónico como "test2@test.com". Luego, en los permisos de acceso, si el usuario trabaja en múltiples compañías, las agregamos junto con la predeterminada.

Después, seleccionamos los módulos instalados y definimos los permisos para cada uno. Por ejemplo, en contabilidad, no asignaremos permisos por el momento. Procedemos a la **configuración técnica**, donde marcamos los permisos necesarios. Una vez guardado, el usuario se crea con los permisos predeterminados.

Se muestran los grupos asignados al usuario, junto con los 137 permisos correspondientes. Por ejemplo, dentro del grupo de "permisos extras", se encuentra "creación de contactos", con sus respectivos permisos:

- leer,

- escribir,
- crear,
- o eliminar).

Al visualizar los permisos del usuario "test2", se muestra cada permiso individualmente en lugar del grupo al que pertenecen. Por ejemplo, los permisos para "creación de contactos" incluyen leer, escribir, crear y eliminar.

Si editamos y otorgamos permisos adicionales, como administración de facturación y ventas, los grupos se incrementarán de 8 a 16 debido a los grupos asignados por defecto a estas secciones. Por ejemplo, al agregar permisos de administración en ventas, se incluye el grupo de "ventas administradoras" con permisos específicos.

Los **permisos pueden ser editados** en cualquier momento. Al hacer clic en un permiso, podemos modificarlo, como restringir la capacidad de leer o crear.

También podemos enviar una invitación por correo electrónico al usuario desde este punto.

Finalmente, desde aquí también es posible eliminar al usuario. Al hacerlo, el usuario "test2" ya no estará en la lista de usuarios.

Crear un informe con QWeb

QWeb es el **motor de plantillas** utilizado por Odoo. Es un motor de plantillas **XML** y se utiliza principalmente para generar **fragmentos y páginas HTML**. En el siguiente enlace podréis ver la documentación necesaria para **programar con Qweb**:

[https://www.odoo.com/documentation/13.0/reference/qweb.html#calling-sub-templates.](https://www.odoo.com/documentation/13.0/reference/qweb.html#calling-sub-templates)

Para crear un informe, en primer lugar, identificaremos el **modelo** en el que queremos que aparezca nuestro informe.

En este caso vamos a crear un informe para el módulo de Ventas (`sale.order`), para que aparezca junto al informe estándar de 'presupuesto':

Aprovecharemos el módulo 'modos de pago' del tema anterior, puesto que **el informe QWeb no es más que una vista XML**. De esta forma nos ahorraremos tener que crear un módulo nuevo. Bastará con crear una nueva vista XML, que denominaremos '`informe_custom.xml`', y agregar dicha vista al manifest en la sección data. Ahora informaremos el XML indicando en una primera etiqueta 'report', que se trata de un **tipo de reporte QWEB** y que pertenecerá al **modelo sale.order**.

A continuación, en una segunda etiqueta 'template' es donde crearemos el **contenido del informe**.

Implementación del XML con tipo de reporte Qweb

```
<odoo>
    <data>
        <report id="informe_custom"
            string="Informe Personalizado"
            model="sale.order"
            report_type="qweb-pdf"
            name="modos_pago.informe_custom_template"
        />

        <template id="informe_custom_template">
            <t t-call="web.html_container">
                <div class="header">
                    Esta es la cabecera
                </div>
                <div class="page">
                    <h1>Esto es un informe custom</h1>
                </div>
            </t>
        </template>
    </data>
</odoo>
```

Un **informe Qweb**, por tanto, es un **archivo XML** que contiene una **etiqueta report_type = 'qweb-pdf'**, dentro de un elemento `<report>`, que está a su vez dentro de un elemento `<data>`.

Tras implementar los cambios actualizaremos el módulo y probaremos que el cambio ha surtido efecto.

Analizar informe QWeb desde Odoo

Vamos a ver cómo podemos analizar el código QWeb de un informe existente en Odoo. Tomaremos como ejemplo el formulario "*presupuesto / pedido*" de los pedidos de venta para analizar su código QWeb desde Odoo.

Para ello, seguiremos estos pasos:

1. Nos dirigimos al módulo **Ajustes**.
2. Dentro de "Ajustes", seleccionamos la opción "**Técnico**".
3. Una vez dentro, ingresamos en la sección de "**Acciones en Informes**".
4. En nuestro caso, buscamos el informe de "*presupuesto / pedido*".
5. Hacemos clic en él para abrirlo.
6. Dentro del informe, encontraremos un botón de "**Vista QWeb**".
7. Al hacer clic en este botón, se nos mostrará todo el **código QWeb** asociado a este informe.

Por ejemplo, la vista "report sale order" contiene una llamada en código QWeb al "report sale order document", y dentro de este, encontramos la **estructura del formulario QWeb en Odoo**.

También es posible buscar este formulario QWeb en una de las vistas del módulo **desde PyCharm**, donde encontraremos el **mismo código**.

Internal Server Error

Planteamiento: Al abrir en el navegador el <http://localhost:8069/> se produce un **error** de

'Internal Server Error'

y no entra en Odoo.

Nudo: ¿Qué puede estar pasando?

Puede deberse a varios problemas, por ejemplo:

- un **error de código**,
- o que el **servidor** se haya quedado **bloqueado**.

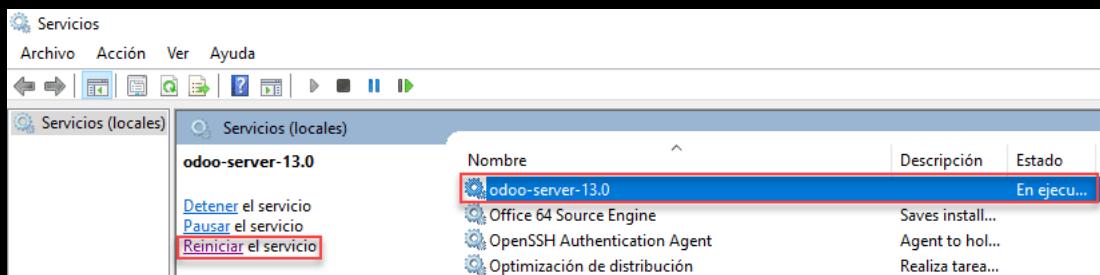
¿Cómo solucionarlo si el servidor se ha quedado bloqueado?

Desenlace: La solución pasa por **reiniciar el servicio**. Para ello, abrimos el '*administrador de tareas*' (cntrl + alt + supr) y en la pestaña 'servicios' buscamos 'odoo-server-13.0' (*la versión que tengamos de Odoo*).

Pulsamos sobre el mismo con el **botón derecho** y a continuación accedemos a '**abrir servicios**'.

Esto nos abrirá una ventana como la que se muestra en la figura, en la que buscaremos el 'odoo-server-13.0' y lo **reiniciaremos**, aunque se esté ejecutando.

Reiniciando el servicio odoo-server-13.0



Si esto no funciona, puede que el error sea debido a:

- **problemas con el usuario**,
- **rol**,
- o **incidencias con el código** que tendremos que solucionar antes de reiniciar el servicio.

Esto lo indicaría el log de errores.

Errores comunes

A continuación, comentaremos 3 errores muy comunes que se dan cuando estamos desarrollando en Odoo y que a la vez son muy fáciles de solucionar:

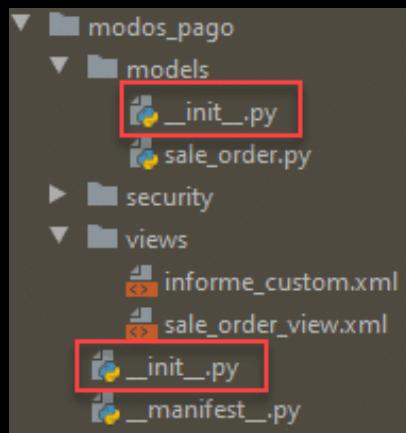
Olivarnos de importar los modelos de Python: y por tanto **no se creen las tablas** en PostgreSQL.

De esta forma, tras crear los modelos e **instalar o actualizar** el módulo **no se vislumbrarían** los cambios.

Es importante recordar que, cada vez que se cree una **carpeta** y en su interior **haya modelos Python** se debe crear un **fichero __init__.py** para **importar** los modelos concretos.

- En el fichero `__init__.py` de la raíz **del módulo** se importará la **carpeta models** que contiene todos los modelos,
- pero en el **__init__.py de la carpeta models** se **importan los modelos** que queremos que realmente se carguen.

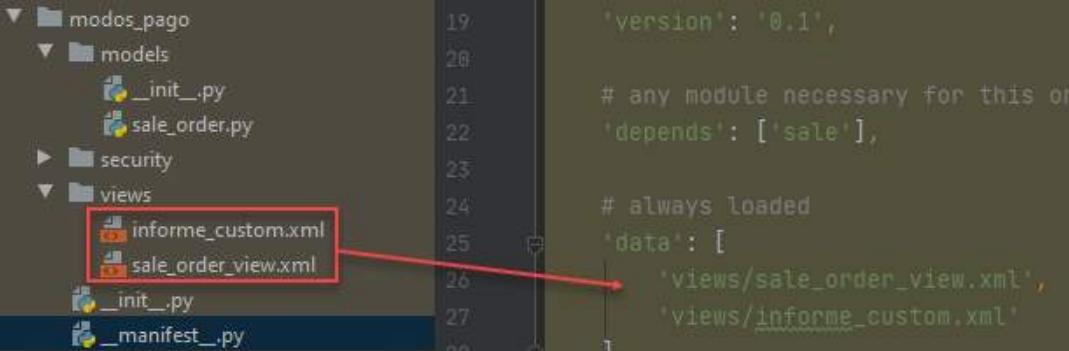
`__init__.py` en la raíz del módulo y en carpeta models



Olivarnos de agregar o importar **en el manifest**, en la **sección 'data'**, la vista (o cualquier **fichero XML**) que hayamos creado **nueva**. Así, al instalar o actualizar el módulo **no se reflejan los cambios**.

Debemos tener en cuenta que, para que una vista se utilice en el módulo, ésta debe estar **informada en la sección 'data'** del manifest. (Interesante el comentario sobre la sección [data] del manifest: '# always loaded').

Informando en el manifest las vistas

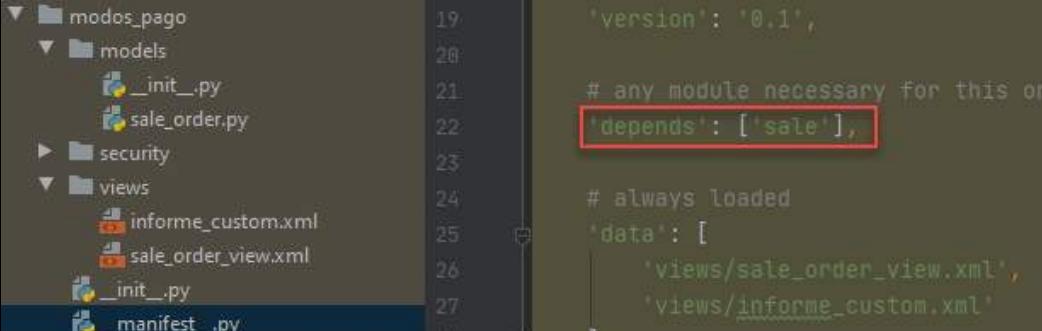


```
19     'version': '0.1',
20
21     # any module necessary for this one
22     'depends': ['sale'],
23
24     # always loaded
25     'data': [
26         'Views/sale_order_view.xml',
27         'views/informe_custom.xml'
28     ]
```

Olvidarnos de indicar las **dependencias** en la sección 'depends' del **manifest** cuando estamos haciendo un módulo que hereda de otro.

Es necesario que indiquemos el/los módulo/s **necesario/s** para que el módulo actual **funcione correctamente**.

Informando en el manifest las dependencias



```
19     'version': '0.1',
20
21     # any module necessary for this one
22     'depends': ['sale'],
23
24     # always loaded
25     'data': [
26         'Views/sale_order_view.xml',
27         'views/informe_custom.xml'
28     ]
```

Los batch-input

Existen opciones para algunos sistemas ERP, que permiten **automatizar determinadas acciones** que necesitemos llevar a cabo en ellos.

Son los denominados batch-input, que permiten **ejecutar transacciones** o programas de forma automática, sin la necesidad de que el usuario interactúe con el programa en sí.

Por ejemplo, en SAP se pueden crear batch-inputs que simulan el proceso de informar campos en la pantalla, pulsar diferentes botones, radiobuttons, checkbox, etc. tal y como lo haría un usuario, solo que de manera automática. De esta forma, el usuario o el sistema ejecutaría el batch-input y éste se encargaría de ir **informando campos y pasando** por las diferentes **pantallas** de **forma automática**.

Los batch-inputs pueden ejecutarse en primer plano (mientras el usuario ve como se rellenan los campos y se van pasando por las pantallas), pero **lo normal** es que se ejecute **en segundo plano**.

Reflexión

Gracias a la gestión de permisos de los usuarios es posible **acotar en Odoo el acceso a los módulos** por parte de los trabajadores de una empresa. Para ello, lo ideal sería **crear un grupo para cada departamento** con los permisos que cada uno de ellos deba tener, y a continuación, asignar dicho grupo a todos los **usuarios del departamento**.

De esta forma, si entra un **empleado nuevo** en un departamento bastará con **agregarle el grupo** y ya tendrá los mismos **permisos** que los usuarios de su departamento. Cualquier **modificación sobre dicho grupo** afectará a **todos los empleados** del departamento.

Si un empleado perteneciese a varios departamentos, simplemente habría que agregarle los grupos de dichos departamentos para que tenga todos los permisos de esos grupos.

<https://www.odoo.com>

<https://praxyformaplus.com/>

<https://odooerpcloud.com/>