

CURSO PASO A PASO

ODOO 20

para Nuevos Programadores



DE CERO A INTEGRACIONES · 26 CAPÍTULOS

Marlon Falcón Hernández

Valencia · 2026

Odoo 20 para Nuevos Programadores

Curso paso a paso de desarrollo de módulos, de cero a integraciones

Autor: Marlon Falcón Hernández

ERP, CRM y Software · Valencia, España · marlonfalcon.com

Edición para Odoo 20.0 · Octubre de 2026

Basado en los ejercicios del repositorio [nuevos-programadores](#)

Índice

Introducción

Instalar Odoo 20 en Windows y Mac

Qué es Odoo y cómo está construido

Capítulo 0 · Python básico para Odoo

Parte I · Fundamentos

Capítulo 1 · Hola mundo: tu primer módulo

Capítulo 2 · Añadir, ocultar y eliminar campos

Capítulo 3 · Campos calculados

Capítulo 4 · Traducciones

Capítulo 5 · Datos iniciales y de demostración

Capítulo 6 · Secuencias

Capítulo 7 · Documento con líneas y pestañas

Capítulo 8 · Varios teléfonos por contacto y botón inteligente

Parte II · Seguridad y búsqueda

Capítulo 9 · Permisos: quién puede confirmar ventas y ver precios

Capítulo 10 · Filtros y agrupaciones personalizadas

Parte III · Informes

Capítulo 11 · Informes PDF: personalizar la factura

Capítulo 12 · Modificar el presupuesto de venta

Parte IV · Asistentes y configuración

Capítulo 13 · Asistente desde el menú Acción

Capítulo 14 · Fechas: buscar ventas por rango

Capítulo 15 · Ajustes propios en Configuración

Capítulo 16 · Exportar productos a Excel

Capítulo 17 · Importar productos desde CSV y Excel

Parte V · Contabilidad

Capítulo 18 · Crear una factura desde código

Capítulo 19 · Crear un asiento contable

Capítulo 20 · Factura, pago y conciliación

Parte VI · Integraciones

Capítulo 21 · Leer y escribir en Odoo desde fuera: API JSON-2

Capítulo 22 · Subir ficheros a un FTP con una tarea programada

Capítulo 23 · Tu propia API REST con token

Capítulo 24 · Una página web que busca empleados

Capítulo 25 · Odoo shell: Python directo contra tu base de datos

Parte VII · Odoo e inteligencia artificial

Capítulo 26 · Programar Odoo con Claude

Anexo · Chuleta de migración de Odoo 16 a Odoo 20

Introducción

Este libro convierte los ejercicios del repositorio [nuevos-programadores](#) (escritos para Odoo 16) en un curso paso a paso para **Odoo 20**. Cada capítulo parte de un ejercicio real, explica qué hace, lo escribe de nuevo con la sintaxis de Odoo 20 y termina con una prueba y un reto.

A quién va dirigido

A quien sabe algo de Python pero nunca ha creado un módulo de Odoo. No hace falta saber XML ni SQL: se explica lo necesario en cada capítulo.

Requisitos de Odoo 20

Componente	Versión
Odoo	20.0 (publicado en septiembre de 2026)
Python	3.12 a 3.14
PostgreSQL	16 o superior
Editor	VS Code o PyCharm

Python 3.10 y 3.11 ya no sirven: el código de Odoo 20 usa sintaxis de 3.12.

Preparar el entorno

1. Instala PostgreSQL 16 y crea un usuario para Odoo: `sudo -u postgres createuser -s $USER`.
2. Clona Odoo 20: `git clone --depth 1 -b 20.0 https://github.com/odoo/odoo.git`.
3. Crea un entorno virtual e instala dependencias:

```
python3.12 -m venv venv
source venv/bin/activate
pip install -r odoo/requirements.txt
```

4. Crea una carpeta `mis_modulos/` junto a `odoo/`. Ahí irán todos los ejercicios.
5. Arranca Odoo en modo desarrollo:

```
./odoo/odoo-bin -d curso20 --addons-path=odoo/addons,mis_modulos --dev=xml,reload
```

6. Abre `http://localhost:8069`, entra como `admin / admin` y activa el **modo desarrollador** (Ajustes → Activar modo desarrollador).

Cómo instalar cada ejercicio

Cada vez que crees un módulo nuevo: Aplicaciones → *Actualizar lista de aplicaciones* → busca el módulo → *Activar*. Cuando cambies código Python, reinicia Odoo. Cuando cambies XML, actualiza el módulo con `-u nombre_modulo`.

Anatomía de un módulo

```
mi_modulo/
├── __init__.py          # importa la carpeta models
├── __manifest__.py      # nombre, versión, dependencias, ficheros de datos
```

```

├─ models/           # clases Python (la lógica y los campos)
├─ views/            # vistas XML (formularios, listas, menús)
├─ security/
│   └─ ir.access.csv # permisos (NUEVO en Odoo 20)
├─ data/             # datos iniciales y secuencias
├─ report/           # informes PDF
├─ wizard/           # asistentes (ventanas emergentes)
├─ controllers/      # rutas web y API
└─ i18n/             # traducciones (.po)

```

Lo que más cambia respecto al repositorio original

- **Seguridad:** `ir.model.access.csv` desaparece. En Odoo 20 los permisos y las reglas de registro se unifican en `ir.access.csv` (capítulo 1 y capítulo 9).
- **Vistas:** la etiqueta `<tree>` es ahora `<list>` y `view_mode` usa `list,form`. El atributo `attrs` ya no existe: se escribe `invisible="..."` directamente.
- **Creación de registros:** `create` recibe una lista de diccionarios con `@api.model_create_multi`.
- **Integraciones:** XML-RPC sigue funcionando pero está obsoleto; lo nuevo es la API **JSON-2** con clave `API` (capítulo 21).
- **Parámetros:** `get_param` se sustituye por métodos tipados como `get_int` o `get_bool` (capítulo 15).

Al final del libro hay una chuleta con todos estos cambios.

Instalar Odoo 20 en Windows y Mac

Hay tres caminos. Para este curso el recomendado es **Docker**: funciona igual en Windows y Mac, y tus módulos viven en una carpeta normal de tu ordenador.

Método	Windows	Mac	Ventaja	Inconveniente
Instalador oficial	Sí	No existe	Un doble clic, incluye PostgreSQL	Solo Community; incómodo para desarrollar
Docker	Sí	Sí	Igual en todos los sistemas, fácil de borrar y repetir	Hay que instalar Docker Desktop
Código fuente	Sí	Sí	Control total, depuración con el editor	Más pasos (ver Introducción)

Opción A · Instalador en Windows

1. Descarga el instalador de **Odoo 20.0 Community** desde nightly.odoo.com/20.0/nightly/windows (archivo `odoo_20.0.latest.exe`).
2. Ejecútalo como administrador y elige **Odoo Server + PostgreSQL**. El instalador incluye su propio Python y PostgreSQL.
3. Acepta la ruta por defecto (`C:\Program Files\Odoo 20.0...`) y deja el puerto **8069**.
4. Al terminar, Odoo queda instalado como servicio de Windows y arranca solo. Abre `http://localhost:8069` y crea la base de datos `curso20`.

Para usar tus propios módulos:

1. Crea `C:\odoo\mis_modulos`.
2. Abre como administrador `C:\Program Files\Odoo 20.0...\server\odoo.conf` y añade la carpeta a `addons_path`:

```
addons_path = C:\Program Files\Odoo 20.0.xxxx\server\odoo\addons,C:\odoo\mis_modulos
```

3. Reinicia el servicio: tecla Windows → *Servicios* → **odoo-server-20.0** → Reiniciar.

Los registros de errores están en `...\server\odoo.log`.

Opción B · Mac sin instalador

Odoo no publica instalador para macOS. Usa Docker (opción C) o el código fuente con Homebrew:

```
# 1. Herramientas
xcode-select --install
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
brew install python@3.12 postgresql@16 git wkhtmltopdf
brew services start postgresql@16

# 2. Código y entorno
git clone --depth 1 -b 20.0 https://github.com/odoo/odoo.git
python3.12 -m venv venv && source venv/bin/activate
pip install -r odoo/requirements.txt

# 3. Arrancar
mkdir mis_modulos
./odoo/odoo-bin -d curso20 --addons-path=odoo/addons,mis_modulos --dev=xml,reload
```

Si `pip` falla compilando `psycpg2`, ejecuta antes `export PATH="$(brew --prefix postgresql@16)/bin:$PATH"`.

Opción C · Docker en Windows y Mac (recomendada)

1. Instala [Docker Desktop](#). En Windows activa WSL 2 cuando te lo pida. En Mac elige la versión Apple Silicon o Intel según tu equipo.
2. Crea una carpeta `curso-odoo20` con esta estructura:

```
curso-odoo20/  
├── docker-compose.yml  
├── config/  
│   └── odoo.conf  
└── addons/      ← aquí irán tus módulos del curso
```

3. `docker-compose.yml`:

```
services:  
  db:  
    image: postgres:16  
    environment:  
      POSTGRES_DB: postgres  
      POSTGRES_USER: odoo  
      POSTGRES_PASSWORD: odoo  
    volumes:  
      - db-data:/var/lib/postgresql/data  
  
  odoo:  
    image: odoo:20.0  
    depends_on:  
      - db  
    ports:  
      - "8069:8069"  
    environment:  
      HOST: db  
      USER: odoo  
      PASSWORD: odoo  
    volumes:  
      - odoo-data:/var/lib/odoo  
      - ./config:/etc/odoo  
      - ./addons:/mnt/extra-addons  
    command: --dev=xml,reload  
  
volumes:  
  odoo-data:  
  db-data:
```

4. `config/odoo.conf`:

```
[options]  
addons_path = /mnt/extra-addons  
admin_passwd = cambia_esta_clave
```

5. Arranca desde una terminal dentro de la carpeta:

```
docker compose up -d  
docker compose logs -f odoo      # ver el log en directo (Ctrl+C para salir)
```

6. Abre `http://localhost:8069` y crea la base de datos `curso20`.

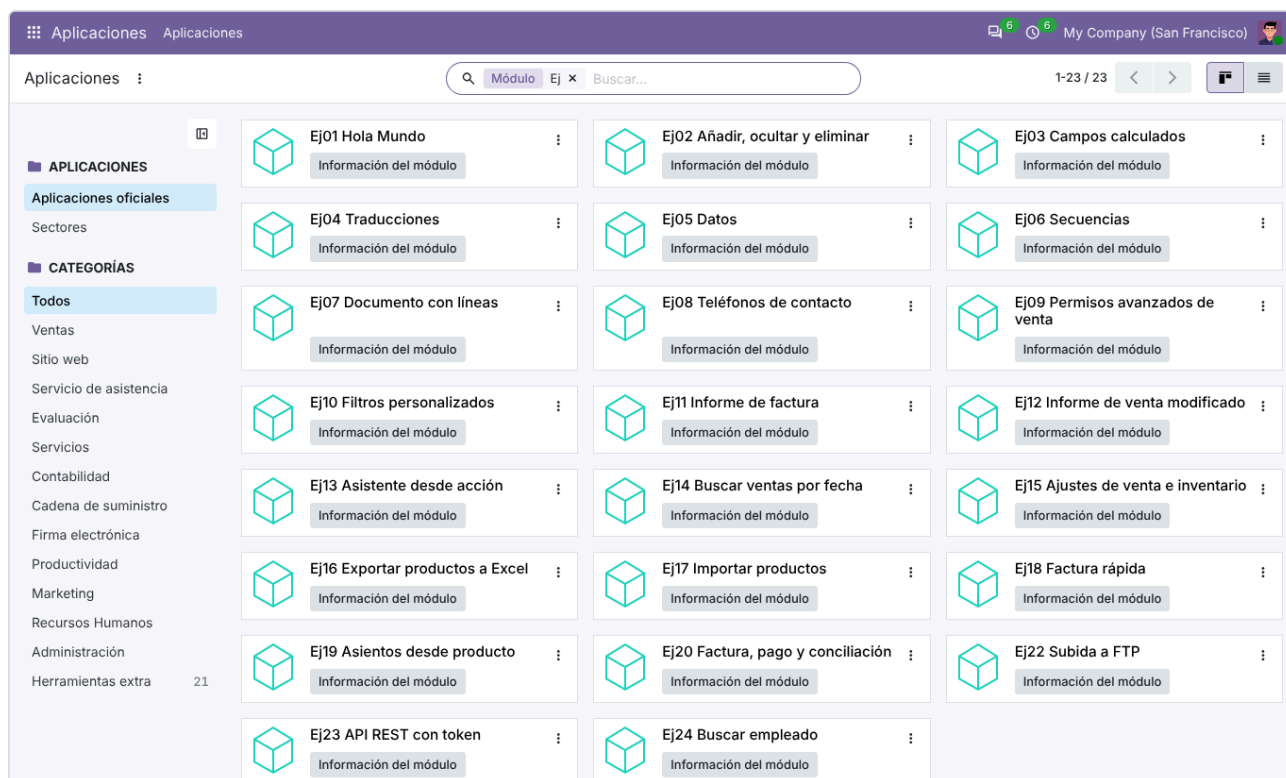
Comandos que usarás a diario:

Tarea	Comando
Parar	<code>docker compose stop</code>
Reiniciar Odoo tras cambiar Python	<code>docker compose restart odoo</code>
Actualizar un módulo	<code>docker compose exec odoo odoo -d curso20 -u ej01_hello_world --stop-after-init --db_host=db --db_user=odoo --db_password=odoo</code>
Abrir el shell de Odoo	<code>docker compose exec odoo odoo shell -d curso20 --db_host=db --db_user=odoo --db_password=odoo</code>
Borrarlo todo y empezar de cero	<code>docker compose down -v</code>

Comprueba que todo funciona

Copia la carpeta `ej01_hello_world` del capítulo 1 dentro de tu carpeta de módulos (`mis_modulos` o `addons`), reinicia Odoo, actualiza la lista de aplicaciones e instálalo. Si aparece el menú **Mascotas**, tu entorno está listo.

Así se ve Aplicaciones con todos los módulos del libro instalados en Odoo 20 (filtro *Ej*):



Qué es Odoo y cómo está construido

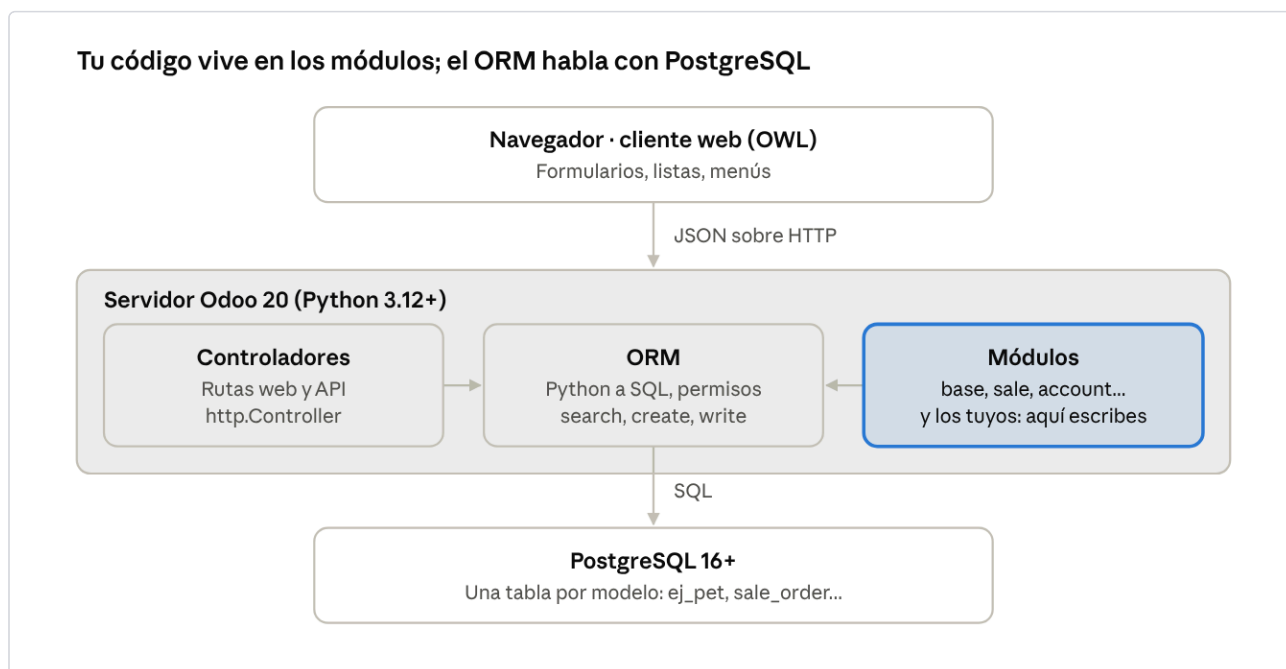
Odoo es un ERP de código abierto: un único programa que reúne ventas, compras, inventario, contabilidad, RR. HH., web y decenas de áreas más, todas sobre la misma base de datos. Cada área es un **módulo** que se instala o no según la empresa. Programar en Odoo es, casi siempre, crear módulos que añaden o cambian comportamiento sin tocar el código original.

Ediciones

Edición	Licencia	Incluye
Community	LGPL-3, gratuita	El núcleo y las aplicaciones básicas (ventas, compras, inventario, facturación)
Enterprise	De pago, por usuario	Community + aplicaciones extra (contabilidad completa, firma, estudio, móvil) y soporte
Odoo Online / Odoo.sh	Servicio en la nube	Enterprise alojado por Odoo; Odoo.sh admite módulos propios

Todo lo de este libro funciona en Community.

Las capas



Arquitectura de Odoo · 3 capas

El navegador solo dibuja: toda la lógica corre en el servidor. Los módulos (los de Odoo y los tuyos) definen modelos, vistas y reglas; el ORM los convierte en consultas SQL y comprueba permisos antes de tocar PostgreSQL.

Las piezas, una a una

Pieza	Qué es	Dónde lo verás
PostgreSQL	La base de datos. Cada modelo es una tabla	Capítulo 1 (la tabla <code>ej_pet</code>)
ORM	Capa Python que traduce objetos a SQL: <code>search</code> , <code>create</code> , <code>write</code> , <code>unlink</code>	Todos los capítulos
Modelo (<code>models.Model</code>)	Una clase Python = una tabla + su lógica	Capítulo 1
Vista (<code>ir.ui.view</code>)	XML que describe formularios, listas, kanban, búsqueda	Capítulos 1 y 2
Acción (<code>ir.actions.*</code>)	Qué pasa al pulsar un menú o botón: abrir vista, imprimir, ejecutar código	Capítulos 1 y 13

Pieza	Qué es	Dónde lo verás
Menú (<code>ir.ui.menu</code>)	Árbol de navegación que lanza acciones	Capítulo 1
Seguridad (<code>res.groups</code> , <code>ir.access</code>)	Quién puede leer, crear, modificar o borrar	Capítulos 1, 8 y 9
QWeb	Motor de plantillas para PDF y páginas web	Capítulos 11 y 12
Controlador (<code>http.Controller</code>)	Rutas HTTP: páginas web y APIs	Capítulos 23 y 24
Cliente web (OWL)	La interfaz en el navegador, escrita en JavaScript	No se toca en este curso

El ciclo de una petición

Cuando un usuario abre la lista de mascotas ocurre esto:

1. El navegador pide la acción del menú **Mascotas**.
2. El servidor devuelve la acción y las vistas `list` y `form` en XML.
3. El cliente web pide los datos: llama a `web_search_read` sobre `ej.pet`.
4. El ORM comprueba los permisos de `ir.access`, construye el SQL y consulta PostgreSQL.
5. El resultado vuelve como JSON y el cliente lo dibuja.

Tú solo escribes los pasos de los extremos: el modelo (Python) y la vista (XML). Lo demás lo hace Odoo.

Herencia: la idea clave

Odoo se adapta **sin modificar** su código. Un módulo nuevo declara `_inherit = 'sale.order'` y añade campos o cambia métodos; una vista nueva declara `inherit_id` y añade o quita elementos. Al actualizar Odoo, tus cambios siguen ahí porque viven en tu módulo. Por eso la primera regla es: **nunca edites los ficheros de `odoo/addons`**.

Módulos y dependencias

Cada módulo declara en `depends` qué módulos necesita. Odoo los instala en orden: `base` primero, después `mail`, `product`, `sale` ... y al final el tuyo. Si tu módulo amplía las ventas, debe depender de `sale`.

Entornos: `self.env`

Dentro de cualquier método tienes `self.env`, el entorno de ejecución:

- `self.env['res.partner']` da acceso a cualquier modelo.
- `self.env.user` y `self.env.company` son el usuario y la empresa actuales.
- `self.env.context` lleva datos extra (idioma, valores por defecto).
- `self.env.cr` es el cursor de la base de datos; casi nunca lo usarás directamente.

Versiones

Odoo publica una versión mayor al año, en otoño. Odoo 20 salió en septiembre de 2026. Cada versión cambia APIs, y un módulo escrito para la 16 casi nunca funciona sin cambios en la 20: por eso este libro reescribe los ejercicios originales.

Capítulo 0 · Python básico para Odoo

No necesitas dominar Python para empezar con Odoo: basta con lo que aparece en este capítulo. Todo lo que verás aquí sale después en los ejercicios.

Variables y tipos

```
nombre = 'Coco'          # str (texto)
edad = 2                  # int (entero)
precio = 19.95            # float (decimal)
activo = True             # bool (True / False)
nada = None               # ausencia de valor
```

En Odoo, un campo vacío no vale `None` sino `False`. Por eso verás mucho `if record.color:` en vez de `if record.color is not None:`.

Texto con f-strings

```
mensaje = f'{nombre} tiene {edad} años'
# 'Coco tiene 2 años'
```

Listas, tuplas y diccionarios

```
colores = ['negro', 'blanco']      # lista: ordenada y modificable
colores.append('marrón')

dominio = [('age', '>', 1)]          # lista de tuplas: así son los filtros de Odoo

vals = {'name': 'Coco', 'age': 2}   # diccionario: clave -> valor
vals['color'] = 'negro'
```

Los diccionarios son lo más importante: con ellos se crean y modifican registros (`create(vals)` , `write(vals)`).

Condiciones y bucles

```
if edad >= 10:
    tipo = 'mayor'
elif edad >= 2:
    tipo = 'adulto'
else:
    tipo = 'cachorro'

for color in colores:
    print(color)

for i, color in enumerate(colores, start=1):
    print(i, color)
```

Python usa la **sangría** (4 espacios) para marcar bloques. Un espacio de más o de menos es un error.

Comprensiones de listas

```
edades = [1, 4, 7]
dobles = [e * 2 for e in edades]      # [2, 8, 14]
mayores = [e for e in edades if e > 3] # [4, 7]
```

Funciones

```
def saludar(nombre, saludo='Hola'):
    return f'{saludo}, {nombre}'

saludar('Coco')           # 'Hola, Coco'
saludar('Coco', saludo='Hey') # 'Hey, Coco'
```

`saludo='Hola'` es un valor por defecto; `saludo='Hey'` es un argumento con nombre. Odoo usa mucho ambos.

Clases y herencia

En Odoo cada modelo es una clase. Esto es lo mínimo que hay que entender:

```
class Animal:
    def __init__(self, nombre):
        self.nombre = nombre        # atributo de la instancia

    def describir(self):
        return f'Soy {self.nombre}'

class Perro(Animal):                # Perro hereda de Animal
    def describir(self):
        texto = super().describir() # llama al método del padre
        return texto + ' y ladro'

Perro('Coco').describir()           # 'Soy Coco y ladro'
```

- `self` es el propio objeto. En Odoo, `self` es un **conjunto de registros** (*recordset*), que puede contener cero, uno o muchos.
- `super()` llama a la versión original del método. En Odoo se usa siempre al sobrescribir `create`, `write` o `unlink` para no romper el comportamiento estándar.

Decoradores

Una línea con `@` encima de una función le añade comportamiento. En Odoo verás:

Decorador	Para qué sirve
<code>@api.depends('campo')</code>	Recalcula un campo calculado cuando cambia <code>campo</code>
<code>@api.onchange('campo')</code>	Reacciona en el formulario antes de guardar
<code>@api.constrains('campo')</code>	Valida datos al guardar
<code>@api.model_create_multi</code>	Sobrescribe <code>create</code> recibiendo una lista de diccionarios

Excepciones

```
from odoo.exceptions import UserError, ValidationError

if edad < 0:
    raise ValidationError('La edad no puede ser negativa.')

try:
    valor = int('abc')
except ValueError:
    valor = 0
```

`UserError` y `ValidationError` muestran un mensaje al usuario y deshacen la operación.

Importaciones y paquetes

```
from odoo import api, fields, models # lo que usarás en cada modelo
from datetime import date, timedelta
```

Una carpeta con `__init__.py` es un **paquete**. Por eso cada módulo de Odoo tiene `__init__.py` que importa sus subcarpetas: si olvidas una línea `from . import ...`, Odoo no verá ese fichero.

Fechas

```
from datetime import date, timedelta

hoy = date.today()
dentro_de_una_semana = hoy + timedelta(days=7)
hoy.strftime('%d/%m/%Y') # '30/10/2026'
```

Odoo trae ayudas propias: `fields.Date.today()` y `fields.Datetime.now()`.

Del Python puro al ORM de Odoo

Python que ya conoces	Equivalente en Odoo
<code>vals = {...}</code>	<code>self.env['ej.pet'].create(vals)</code>
Filtrar una lista	<code>self.env['ej.pet'].search([('age', '>', 1)])</code>
<code>objeto.atributo = x</code>	<code>record.write({'campo': x})</code> o <code>record.campo = x</code>
<code>for x in lista:</code>	<code>for record in self:</code>
<code>[x.nombre for x in lista]</code>	<code>records.mapped('name')</code>
<code>[x for x in lista if ...]</code>	<code>records.filtered(lambda r: r.age > 1)</code>
<code>len(lista)</code>	<code>len(records)</code> o <code>search_count(dominio)</code>

`self.env` es la puerta de entrada a todo: `self.env['modelo']` devuelve el modelo, `self.env.user` el usuario actual y `self.env.company` la empresa activa.

Practica antes de seguir

Abre una consola con `python3` y comprueba que entiendes este fragmento:

```
mascotas = [
    {'name': 'Coco', 'age': 2},
    {'name': 'Pinki', 'age': 1},
    {'name': 'Rex', 'age': 9},
]
adultos = [m['name'] for m in mascotas if m['age'] >= 2]
print(f'Adultos: {", ".join(adultos)}')
# Adultos: Coco, Rex
```

Si sabes qué hace cada línea, estás listo para el capítulo 1.

Parte I · Fundamentos

Capítulo 1 · Hola mundo: tu primer módulo

Al terminar tendrás una aplicación **Mascotas** con su propio menú, una lista y un formulario. Ejercicio original: `01_hello_world`.

Nombre del módulo. En el repositorio las carpetas empiezan por número (`01_hello_world`). Funciona, pero un identificador Python no debe empezar por dígito. En este libro usaremos `ej01_hello_world`, `ej02...`, etc.

Antes de empezar · Modelo, vista, acción y menú

Todo lo que ves en Odoo se construye con cuatro piezas. Este capítulo crea una de cada:

Pieza	Qué es	Dónde se escribe	En este capítulo
Modelo	Los datos y su lógica. Una clase Python que Odoo convierte en una tabla de PostgreSQL	<code>models/*.py</code>	<code>ej.pet</code> → tabla <code>ej_pet</code>
Vista	Cómo se muestran esos datos en pantalla: formulario, lista, kanban, búsqueda...	<code>views/*.xml</code> (<code>ir.ui.view</code>)	Un formulario y una lista
Acción	Qué ocurre al pulsar algo: abrir un modelo con unas vistas concretas	<code>views/*.xml</code> (<code>ir.actions.act_window</code>)	Abrir las mascotas en lista y formulario
Menú	La entrada en la barra de navegación que lanza la acción	<code>views/*.xml</code> (<code>menuitem</code>)	Mascotas → Mascotas → Listado

Se encadenan así: el usuario pulsa el **menú** → el menú lanza la **acción** → la acción abre las **vistas** → las vistas muestran los registros del **modelo**.

Una forma fácil de recordarlo: el modelo es la *tabla de Excel*, la vista es el *formato* con el que la miras, la acción es el *botón* que la abre y el menú es el *sitio* donde está ese botón.

Un modelo puede tener muchas vistas (varias listas, un kanban, un formulario simple y otro completo), y el mismo modelo puede abrirse desde varios menús con acciones distintas (por ejemplo, *Todos los contactos* y *Solo empresas*).

Antes de empezar · Tipos de campo de Odoo

Cada atributo `fields.X` de un modelo es una columna de la tabla. El tipo decide qué se guarda, cómo se ve en pantalla y qué columna crea en PostgreSQL.

Campos simples

Campo	Guarda	Valor en Python	Columna PostgreSQL	Ejemplo
Char	Texto corto (una línea)	<code>str</code>	<code>varchar</code>	<code>name = fields.Char()</code>
Text	Texto largo (varias líneas)	<code>str</code>	<code>text</code>	<code>notes = fields.Text()</code>
Html	Texto con formato (negrita, listas, imágenes)	<code>str</code> (HTML)	<code>text</code>	<code>description = fields.Html()</code>
Integer	Número entero	<code>int</code>	<code>int4</code>	<code>age = fields.Integer()</code>

Campo	Guarda	Valor en Python	Columna PostgreSQL	Ejemplo
Float	Número decimal	float	numeric	<code>weight = fields.Float(digits=(6, 2))</code>
Monetary	Importe en una moneda	float	numeric	<code>price = fields.Monetary(currency_field='currency_id')</code>
Boolean	Sí / no	bool	bool	<code>vaccinated = fields.Boolean()</code>
Date	Fecha	datetime.date	date	<code>birth_date = fields.Date()</code>
Datetime	Fecha y hora (se guarda en UTC)	datetime.datetime	timestamp	<code>last_visit = fields.Datetime()</code>
Selection	Una opción de una lista fija	str (la clave)	varchar	<code>type = fields.Selection([('small', 'Pequeña'), ('big', 'Grande')])</code>
Binary	Un fichero	Objeto binario (<code>.content</code> da los bytes)	Adjunto	<code>document = fields.Binary()</code>
Image	Una imagen (redimensionada automáticamente)	Objeto binario	Adjunto	<code>photo = fields.Image(max_width=512, max_height=512)</code>

`Monetary` necesita un campo `currency_id = fields.Many2one('res.currency')` en el mismo modelo para saber en qué moneda está el importe.

Campos relacionales (enlazan un registro con otros)

Campo	Relación	Ejemplo	Lo verás en
Many2one	Este registro apunta a uno de otro modelo	<code>owner_id = fields.Many2one('res.partner', string='Dueño')</code>	Capítulos 7,8
One2many	Este registro tiene muchos del otro modelo (el inverso de un <code>Many2one</code>)	<code>line_ids = fields.One2many('main.model.line', 'main_model_id')</code>	Capítulos 7,8
Many2many	Muchos con muchos	<code>tag_ids = fields.Many2many('res.partner.category')</code>	Capítulo 13

Por convención, los `Many2one` terminan en `_id` y los `One2many` y `Many2many` en `_ids`.

Cuidado con los valores vacíos: en Odoo un campo sin valor no vale `None`, vale `False` (y un relacional vacío es un conjunto vacío). Por eso se comprueba con `if record.color:`.

Antes de empezar · Atributos comunes de los campos

Todos los campos aceptan estos parámetros:

Atributo	Para qué sirve	Ejemplo
string	Etiqueta que ve el usuario	<code>string='Nombre'</code>
required	Obligatorio: no se puede guardar vacío	<code>required=True</code>
readonly	Solo lectura en la interfaz	<code>readonly=True</code>
default	Valor inicial al crear	<code>default='small'</code> o <code>default=lambda self: self.env.user</code>
help	Texto de ayuda al pasar el ratón	<code>help='Edad en años'</code>
index	Crea un índice en la base de datos para buscar más rápido	<code>index=True</code>
copy	Si se copia al duplicar el registro (por defecto sí, salvo relacionales <code>One2many</code>)	<code>copy=False</code>
tracking	Registra los cambios en el historial (<i>chatter</i>); el modelo debe heredar <code>mail.thread</code>	<code>tracking=True</code>
compute	Valor calculado por un método	<code>compute='_compute_total'</code> (capítulo 3)
related	Copia el valor de un campo de otro registro enlazado	<code>related='owner_id.email'</code>
groups	Solo visible para ciertos grupos	<code>groups='base.group_system'</code>

Con un ejemplo completo:

```
weight = fields.Float(
    string='Peso (kg)',
    digits=(6, 2),
    required=True,
    default=1.0,
    help='Peso de la mascota en kilogramos',
    tracking=True,
)
```

No hace falta memorizar estas tablas: vuelve aquí cada vez que necesites un tipo de campo nuevo. En el reto de este capítulo usarás `Date`, y al final del capítulo 7 ya habrás usado casi todos.

Paso 1 · Crea la carpeta y el manifiesto

Dentro de `mis_modulos/` crea `ej01_hello_world/` con este `__manifest__.py`:

```
{
    'name': 'Ej01 Hola Mundo',
    'version': '20.0.1.0.0',
    'author': 'Tu nombre',
    'license': 'LGPL-3',
    'category': 'Extra Tools',
    'summary': 'Primer módulo: gestión de mascotas',
    'depends': ['base'],
    'data': [
        'security/ir.access.csv',
        'views/ej_pet_views.xml',
    ],
    'application': True,
}
```

La versión empieza por `20.0`: así Odoo sabe para qué versión es. El orden de `data` importa: primero la seguridad, después las vistas.

Paso 2 · Crea el modelo

`__init__.py` (raíz):

```
from . import models
```

`models/__init__.py`:

```
from . import ej_pet
```

`models/ej_pet.py`:

```
from odoo import fields, models

class EjPet(models.Model):
    _name = 'ej.pet'
    _description = 'Mascota'

    name = fields.Char(string='Nombre', required=True)
    age = fields.Integer(string='Edad')
    color = fields.Char(string='Color')
    type = fields.Selection(
        [('small', 'Pequeña'), ('medium', 'Mediana'), ('big', 'Grande')],
```

```

        string='Tamaño', default='small', required=True,
    )

```

- `_name` es el nombre técnico. Odoo crea la tabla `ej_pet` en PostgreSQL.
- `_description` es obligatorio en la práctica: sin él Odoo muestra un aviso en el log.
- Cada atributo `fields.X` se convierte en una columna.

Paso 3 · Da permisos (cambio de Odoo 20)

En Odoo 16 se usaba `ir.model.access.csv`. En Odoo 20 ese modelo ya no existe; se usa `security/ir.access.csv`:

```

id,name,model_id,group_id/id,operation,domain
access_ej_pet_user,ej.pet usuario,ej.pet,base.group_user,crud,

```

Columna	Significado
<code>model_id</code>	Nombre técnico del modelo, tal cual (<code>ej.pet</code>)
<code>group_id/id</code>	Grupo que recibe el permiso (<code>base.group_user</code> = usuarios internos)
<code>operation</code>	Letras <code>c</code> crear, <code>r</code> leer, <code>u</code> actualizar, <code>d</code> borrar
<code>domain</code>	Opcional: limita a qué registros se aplica

Cuidado: en el original la columna de grupo iba vacía. En Odoo 20 una fila **sin grupo** no da permiso: es una *restricción* para todos. Pon siempre un grupo.

Paso 4 · Crea las vistas y el menú

`views/ej_pet_views.xml`:

```

<?xml version="1.0" encoding="UTF-8"?>
<odoo>
  <record id="view_ej_pet_form" model="ir.ui.view">
    <field name="name">ej.pet.form</field>
    <field name="model">ej.pet</field>
    <field name="arch" type="xml">
      <form string="Mascota">
        <sheet>
          <group>
            <group>
              <field name="name"/>
              <field name="age"/>
            </group>
            <group>
              <field name="color"/>
              <field name="type"/>
            </group>
          </group>
        </sheet>
      </form>
    </field>
  </record>

  <record id="view_ej_pet_list" model="ir.ui.view">
    <field name="name">ej.pet.list</field>
    <field name="model">ej.pet</field>
    <field name="arch" type="xml">
      <list>
        <field name="name"/>
        <field name="age"/>
        <field name="color" optional="show"/>
        <field name="type"/>
      </list>
    </field>
  </record>
</odoo>

```

```

        </list>
    </field>
</record>

<record id="act_ej_pet" model="ir.actions.act_window">
    <field name="name">Mascotas</field>
    <field name="res_model">ej.pet</field>
    <field name="view_mode">list,form</field>
</record>

<menuitem id="ej_pet_menu_root" name="Mascotas" sequence="10"/>
<menuitem id="ej_pet_menu_main" name="Mascotas" parent="ej_pet_menu_root" sequence="10"/>
<menuitem id="ej_pet_menu_action" name="Listado" parent="ej_pet_menu_main" action="act_ej_pet"
sequence="10"/>
</odoo>

```

Diferencias con el original: `<tree>` pasa a `<list>`, `view_mode` pasa a `list,form` y el formulario va dentro de `<sheet>` para que tenga el aspecto de hoja.

Paso 5 · Instala y prueba

1. Reinicia Odoo y actualiza la lista de aplicaciones.
2. Instala *Ej01 Hola Mundo*.
3. Entra en el menú **Mascotas** y crea dos registros.

Si ves un error *You are not allowed to access*, revisa el fichero `ir.access.csv`.

Resultado en Odoo 20

El formulario de una mascota creado en este capítulo:

The screenshot shows the Odoo 20 interface for the 'Mascotas' (Pets) model. The header bar is purple with the text 'Mascotas Mascotas' and 'YourCompany' with a user icon. Below the header, there's a 'Nuevo' button and a dropdown menu showing 'Mascotas' and 'Coco'. The main content area displays a form with the following fields:

Nombre	Coco	Color	Azul
Edad	2	Tamaño	Pequeña

At the bottom right of the form, there is a dropdown arrow.

Reto

Añade un campo `birth_date = fields.Date(string='Nacimiento')` y muéstralo en la lista y el formulario.

Capítulo 2 · Añadir, ocultar y eliminar campos

Nunca se modifica el código de otro módulo: se **hereda**. En este capítulo un segundo módulo amplía las mascotas del capítulo 1 sin tocarlo. Ejercicio original: 02_add_hide_delete.

Paso 1 · Manifiesto con dependencia

```
{
    'name': 'Ej02 Añadir, ocultar y eliminar',
    'version': '20.0.1.0.0',
    'license': 'LGPL-3',
    'depends': ['ej01_hello_world'],
    'data': ['views/ej_pet_views.xml'],
}
```

`depends` garantiza que el módulo 1 se instala antes y que sus vistas existen.

Paso 2 · Herencia del modelo

`models/ej_pet.py`:

```
from odoo import fields, models

class EjPet(models.Model):
    _inherit = 'ej.pet'

    pretty_name = fields.Boolean(string='Nombre bonito')
    my_age = fields.Integer(string='Mi edad')
```

Con `_inherit` y `sin _name`, Odoo añade las columnas a la tabla existente `ej_pet`. No hace falta un nuevo `ir.access.csv`: el permiso del modelo ya existe.

Paso 3 · Herencia de la vista

Una vista heredada no repite todo el formulario; solo describe los cambios con `position`:

```
<?xml version="1.0" encoding="UTF-8"?>
<odoo>
    <record id="view_ej_pet_form_inherit" model="ir.ui.view">
        <field name="name">ej.pet.form.inherit</field>
        <field name="model">ej.pet</field>
        <field name="inherit_id" ref="ej01_hello_world.view_ej_pet_form"/>
        <field name="arch" type="xml">
            <!-- AÑADIR: después de name -->
            <field name="name" position="after">
                <field name="pretty_name" widget="boolean_toggle"/>
            </field>
            <!-- SUSTITUIR: age por my_age -->
            <field name="age" position="replace">
                <field name="my_age"/>
            </field>
            <!-- OCULTAR: el campo sigue existiendo -->
            <field name="color" position="attributes">
                <attribute name="invisible">1</attribute>
            </field>
        </field>
    </record>

    <record id="view_ej_pet_list_inherit" model="ir.ui.view">
```

```

<field name="name">ej.pet.list.inherit</field>
<field name="model">ej.pet</field>
<field name="inherit_id" ref="ej01_hello_world.view_ej_pet_list"/>
<field name="arch" type="xml">
    <field name="name" position="after">
        <field name="pretty_name" widget="boolean_toggle"/>
    </field>
</field>
</record>
</odoo>

```

position	Qué hace
after / before	Inserta junto al elemento
inside	Inserta dentro (al final)
replace	Sustituye; vacío = elimina de la vista
attributes	Cambia atributos (invisible, readonly, required, string)

Ocultar con condición (cambio desde Odoo 17)

El antiguo `attrs="{ 'invisible': [...] }"` ya no existe. Ahora la condición es una expresión Python directa:

```

<field name="my_age" invisible="not pretty_name"/>
<field name="color" readonly="type == 'big'"/>

```

Eliminar o esconder

`replace` vacío borra el campo de la vista, pero si otra vista heredada lo busca, fallará. Ocultar con `invisible` es más seguro. Nunca elimines un campo obligatorio sin valor por defecto: no se podría guardar el registro.

Prueba

Instala *Ej02*, abre una mascota y comprueba: aparece el interruptor; *Edad* se ha convertido en *Mi edad* y *Color* no se ve.

Resultado en Odoo 20

El interruptor *Nombre bonito* aparece en la lista:

Mascotas Mascotas					YourCompany 	
Nuevo Mascotas		Buscar...			1-3 / 3 < >	
☐ Nombre	Nombre bonito	Edad	Color	Tamaño		
☐ Coco	☒	2	Azul	Pequeña		
☐ Pinki	☐	1	Negro	Mediana		
☐ Rex	☐	9	Marrón	Grande		

En el formulario, *Edad* se ha sustituido por *Mi edad* y *Color* está oculto:

Mascotas

Mascotas

YourCompany

Nuevo

Mascotas

Coco

1/3

<

>

Nombre

Coco

Tamaño

Pequeña

Nombre bonito

☒

Mi edad

2

Reto

Haz que *Mi edad* solo sea visible cuando *Nombre bonito* está activado.

Capítulo 3 · Campos calculados

Un campo calculado obtiene su valor de un método Python en vez de escribirlo el usuario. Ejercicio original: 03_calculated_fields.

Corrección del original. El ejercicio hereda `ej.pets` y la vista `view_ej_pets_form`, que no existen en el módulo 01 (allí el modelo es `ej.pet`). Aquí lo corregimos y heredamos `ej.pet`.

Paso 1 · Manifiesto

```
{
    'name': 'Ej03 Campos calculados',
    'version': '20.0.1.0.0',
    'license': 'LGPL-3',
    'depends': ['ej02_add_hide_delete'],
    'data': ['views/ej_pet_views.xml'],
}
```

Paso 2 · Tres tipos de cálculo

```
from odoo import api, fields, models

class EjPet(models.Model):
    _inherit = 'ej.pet'

    # 1. Calculado al vuelo: no se guarda, no se puede buscar
    is_pretty_name = fields.Boolean(
        string='¿Es bonito?', compute='_compute_is_pretty_name')

    # 2. Calculado y guardado: se puede buscar, ordenar y agrupar
    is_not_pretty_name = fields.Boolean(
        string='¿No es bonito?', compute='_compute_is_not_pretty_name',
        store=True)

    # 3. Valor por defecto calculado con una función
    mydb = fields.Char(
        string='Base de datos', default=lambda self: self.env.cr.dbname)

    @api.depends('pretty_name')
    def _compute_is_pretty_name(self):
        for record in self:
            record.is_pretty_name = record.pretty_name

    @api.depends('pretty_name')
    def _compute_is_not_pretty_name(self):
        for record in self:
            record.is_not_pretty_name = not record.pretty_name
```

Reglas que nunca fallan:

1. El método recorre `self` con `for record in self`: puede recibir muchos registros a la vez.
2. Asigna un valor en **todas** las ramas; si no, Odoo lanza un error.
3. `@api.depends` lista los campos que, al cambiar, obligan a recalcular. Sin él, un campo con `store=True` nunca se actualiza.

Paso 3 · Muéstralo en la vista

```
<odoo>
    <record id="view_ej_pet_form_calc" model="ir.ui.view">
        <field name="name">ej.pet.form.calc</field>
        <field name="model">ej.pet</field>
        <field name="inherit_id" ref="ej01_hello_world.view_ej_pet_form"/>
        <field name="arch" type="xml">
            <field name="name" position="after">
                <field name="is_pretty_name"/>
                <field name="is_not_pretty_name"/>
                <field name="mydb"/>
            </field>
        </field>
    </record>
</odoo>
```

Un ejemplo más útil: edad en meses

```
age_months = fields.Integer(string='Edad (meses)', compute='_compute_age_months', store=True)

@api.depends('age')
def _compute_age_months(self):
    for record in self:
        record.age_months = record.age * 12
```

Novedad desde Odoo 19: `compute_sql`

Si no quieres guardar el campo pero sí poder filtrar por él, puedes darle una expresión SQL con `compute_sql`. Es un tema avanzado; para empezar, usa `store=True`.

Prueba

Activa *Nombre bonito* en una mascota y guarda. ¿Es bonito? cambia al instante; ¿No es bonito? cambia y además aparece al agrupar la lista por ese campo.

Resultado en Odoo 20

Los campos calculados (¿Es bonito?, ¿No es bonito?, Base de datos, Edad (meses)) junto a los del capítulo 2:

The screenshot shows the Odoo 20 interface for the 'Mascotas' (Pets) app. The form is for a pet named 'Coco'. The fields and their values are:

- Nombre: Coco
- Tamaño: Pequeña
- ¿Es bonito?: ☒
- ¿No es bonito?: ☐
- Base de datos: curso_c3
- Edad (meses): 24
- Nombre bonito: ☒
- Mi edad: 0

Reto

Crea un campo calculado `display_label` que devuelva "Coco (2 años)" a partir de nombre y edad.

Capítulo 4 · Traducciones

Odoo se escribe en inglés y se traduce con ficheros `.po`. Así un mismo módulo se ve en español, inglés o francés según el idioma de cada usuario. Ejercicio original: `04_hello_world_translation`.

Paso 1 · Escribe el código en inglés

En este capítulo creamos un modelo independiente, ej.`animal` (menú *Animals*), para ver la traducción desde cero.

Las etiquetas (`string=`) y los textos de las vistas van en inglés. Se traducen después:

```
name = fields.Char(string='Name', required=True)
age = fields.Integer(string='Age')
type = fields.Selection(
    [('small', 'Small'), ('medium', 'Medium'), ('big', 'Big')],
    string='Size', default='small', required=True,
)
```

Paso 2 · Marca los mensajes de Python

Los textos que lanzas desde Python (errores, avisos) deben pasar por la función de traducción. Desde Odoo 18 la forma recomendada es `self.env._()`:

```
from odoo.exceptions import ValidationError

@api.constrains('age')
def _check_age(self):
    for record in self:
        if record.age < 0:
            raise ValidationError(self.env._('Age cannot be negative.'))
```

Para textos con variables, usa marcadores con nombre: `self.env._('%(name)s is too old', name=record.name)`. Nunca construyas el texto con f-string antes de traducirlo.

Paso 3 · Activa el idioma

Ajustes → Idiomas → Activar **Español**. Asigna ese idioma a tu usuario en sus preferencias.

Paso 4 · Exporta la plantilla

Desde la interfaz: Ajustes → Traducciones → Exportar traducción, idioma *Español*, formato *PO*, módulo `ej04_hello_world_translation`. Guarda el fichero como `i18n/es.po` dentro del módulo.

Desde la terminal también se puede con el comando `i18n` de `odoo-bin` (consulta `./odoo-bin i18n --help`).

Paso 5 · Traduce

Cada entrada del `.po` tiene el texto original (`msgid`) y la traducción (`msgstr`):

```
#. module: ej04_hello_world_translation
#: model:ir.model.fields,field_description:ej04_hello_world_translation.field_ej_animal__age
msgid "Age"
msgstr "Edad"

#: model:ir.model.fields.selection,name:ej04_hello_world_translation.selection__ej_animal__type__big
```

```
msgid "Big"
msgstr "Grande"
```

Puedes editarlo con un editor de texto o con **Poedit**. El fichero `.mo` del repositorio original no hace falta: Odoo lee el `.po` directamente.

Paso 6 · Carga la traducción

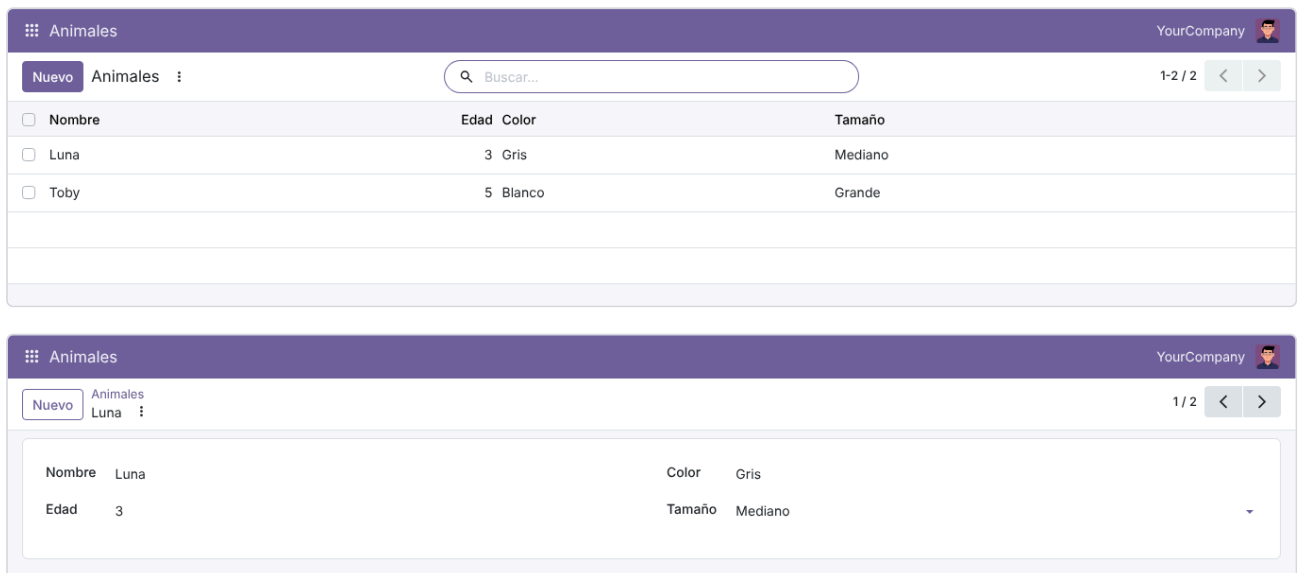
Actualiza el módulo (`-u ej04_hello_world_translation`). Si el idioma ya estaba instalado, en Ajustes → Idiomas pulsa **Actualizar** sobre Español. Cierra sesión y vuelve a entrar.

Prueba

Con tu usuario en español, el formulario muestra *Edad* y *Grande*. Cambia a inglés y vuelve a ver *Age* y *Big*.

Resultado en Odoo 20

El código está en inglés (*Animals*, *Name*, *Size*, *Big*), pero con el usuario en español Odoo muestra las traducciones del `es.po` :



The image displays two screenshots of the Odoo 20 user interface for the 'Animales' (Animals) module, demonstrating the effect of a Spanish translation.

Top Screenshot (List View): The header bar shows 'Animales' and 'YourCompany'. The toolbar includes a 'Nuevo' (New) button, a search bar labeled 'Buscar...', and a pagination indicator '1-2 / 2'. The table lists two animals:

Nombre	Edad	Color	Tamaño
Luna	3	Gris	Mediano
Toby	5	Blanco	Grande

Bottom Screenshot (Form View): The header bar shows 'Animales' and 'YourCompany'. The toolbar includes a 'Nuevo' button and a dropdown menu showing 'Animales Luna'. The form displays the following fields:

Nombre	Luna	Color	Gris
Edad	3	Tamaño	Mediano

Reto

Añade `i18n/fr.po` con la traducción al francés de tres campos.

Capítulo 5 · Datos iniciales y de demostración

Un módulo puede crear registros al instalarse: mascotas de ejemplo, categorías, secuencias o parámetros. Ejercicio original: `05_hello_world_data`.

Paso 1 · Elige entre `data` y `demo`

Clave del manifiesto	Cuándo se carga	Úsalo para
<code>data</code>	Siempre	Datos que el módulo necesita para funcionar (secuencias, categorías, parámetros)
<code>demo</code>	Solo si la base de datos tiene datos de demostración	Registros de ejemplo para probar

Las mascotas de ejemplo son datos de prueba, así que en Odoo 20 lo correcto es ponerlas en `demo`:

```
{
    'name': 'Ej05 Datos',
    'version': '20.0.1.0.0',
    'license': 'LGPL-3',
    'depends': ['ej01_hello_world'],
    'demo': ['data/pet_demo.xml'],
}
```

Paso 2 · Escribe los registros en XML

`data/pet_demo.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<odoo noupdate="1">
    <record id="pet_coco" model="ej.pet">
        <field name="name">Coco</field>
        <field name="age">2</field>
        <field name="color">Azul</field>
        <field name="type">small</field>
    </record>
    <record id="pet_pinki" model="ej.pet">
        <field name="name">Pinki</field>
        <field name="age">1</field>
        <field name="color">Negro</field>
        <field name="type">medium</field>
    </record>
</odoo>
```

- `id` es el **identificador externo** (XML ID). Odoo lo guarda como `ej05_hello_world_data.pet_coco` y así puede encontrar el registro en futuras actualizaciones.
- `noupdate="1"` significa: créalo una vez y no lo vuelvas a sobrescribir al actualizar. Si el usuario cambia el color de Coco, su cambio se respeta.

Paso 3 · Campos relacionales y valores calculados

```
<!-- Many2one: apunta a otro registro por su XML ID -->
<field name="partner_id" ref="base.res_partner_1"/>

<!-- eval: evalúa una expresión Python -->
<field name="age" eval="3"/>
<field name="active" eval="True"/>
```

```
<field name="birth_date" eval="(DateTime.today() - relativedelta(years=2)).strftime('%Y-%m-%d')"/>

<!-- Many2many: añade enlaces -->
<field name="tag_ids" eval="[Command.link(ref('tag_amigable'))]"/>
```

Paso 4 · Instala con datos de demostración

Al crear la base de datos marca *Cargar datos de demostración*. Desde la terminal, en Odoo 20 los datos demo **no** se cargan por defecto; añade `--with-demo` :

```
./odoo/odoo-bin -d curso20_demo --addons-path=odoo/addons,mis_modulos --with-demo -i ej05_hello_world_data
```

Prueba

Abre **Mascotas**: aparecen Coco y Pinki. Activa el modo desarrollador, abre Coco y en el menú de depuración elige *Ver metadatos*: verás su XML ID.

Resultado en Odoo 20

Nada más instalar el módulo en una base con datos de demostración, ya existen Coco, Pinki y Rex:

Mascotas Mascotas				YourCompany 
Nuevo Mascotas		Buscar...		1-3 / 3 < >
☐ Nombre	Edad	Color	Tamaño	⌵
☐ Coco	2	Azul	Pequeña	
☐ Pinki	1	Negro	Mediana	
☐ Rex	9	Marrón	Grande	

Reto

Crea un fichero `data/pet_type_data.xml` (en `data`, no en `demo`) con tres registros de un nuevo modelo `ej.pet.category`.

Capítulo 6 · Secuencias

Cada mascota recibirá un código automático como EJP/2026/00001, igual que las facturas o los pedidos de venta. Ejercicio original: 06_hello_world_sequences.

Paso 1 · Define la secuencia

data/ir_sequence.xml (va en data, porque el módulo la necesita siempre):

```
<?xml version="1.0" encoding="utf-8"?>
<odoo noupdate="1">
  <record id="seq_ej_pet" model="ir.sequence">
    <field name="name">Mascotas</field>
    <field name="code">ej.pet</field>
    <field name="prefix">EJP/%(range_year)s</field>
    <field name="padding">5</field>
    <field name="use_date_range" eval="True"/>
    <field name="company_id" eval="False"/>
  </record>
</odoo>
```

Campo	Efecto
code	Nombre con el que la pides desde Python
prefix	Texto delante del número; %(range_year)s pone el año
padding	Rellena con ceros: 5 → 00001
use_date_range	Reinicia la numeración cada año
company_id = False	La comparten todas las empresas

Paso 2 · Añade el campo y pide el número al crear

```
from odoo import api, fields, models

class EjPet(models.Model):
    _inherit = 'ej.pet'

    code = fields.Char(
        string='Código', default='New', readonly=True, copy=False)

    @api.model_create_multi
    def create(self, vals_list):
        for vals in vals_list:
            if vals.get('code', 'New') == 'New':
                vals['code'] = self.env['ir.sequence'].next_by_code('ej.pet') or 'New'
        return super().create(vals_list)
```

Tres cambios respecto al original:

1. El original escribía `dafault` (error tipográfico): el valor por defecto nunca se aplicaba.
2. `@api.model` con `create(self, vals)` era la forma antigua. En Odoo 20 se usa `@api.model_create_multi`, que recibe una **lista** de diccionarios y permite crear muchos registros en una sola llamada.
3. `super().create(...)` sustituye a `super(Pets, self).create(...)`.

`copy=False` evita que al duplicar una mascota se copie también su código.

Paso 3 · Muestra el código como título

Hereda el formulario y añade el código encima de los grupos:

```
<record id="view_ej_pet_form_seq" model="ir.ui.view">
  <field name="name">ej.pet.form.seq</field>
  <field name="model">ej.pet</field>
  <field name="inherit_id" ref="ej01_hello_world.view_ej_pet_form"/>
  <field name="arch" type="xml">
    <xpath expr="//sheet/group" position="before">
      <div class="oe_title">
        <h1><field name="code"/></h1>
      </div>
    </xpath>
  </field>
</record>
```

`xpath` permite localizar cualquier elemento, no solo campos. `//sheet/group` = el primer `group` dentro de `sheet`.

Paso 4 · Manifiesto

```
'depends': ['ej01_hello_world'],
'data': [
  'data/ir_sequence.xml',
  'views/ej_pet_views.xml',
],
```

Prueba

Crea dos mascotas: obtendrás `EJP/2026/00001` y `EJP/2026/00002`. Puedes cambiar el siguiente número en Ajustes → Técnico → Secuencias.

Resultado en Odoo 20

Las mascotas nuevas reciben su código automático. Fíjate en que las que ya existían antes de instalar el módulo se quedan con `New`: la secuencia solo actúa al crear.

Mascotas Mascotas					YourCompany
Nuevo Mascotas : <input type="text" value="Buscar..."/>					1-5 / 5 < >
☐ Código	Nombre	Edad	Color	Tamaño	⌵
☐ New	Coco	2	Azul	Pequeña	
☐ New	Pinki	1	Negro	Mediana	
☐ New	Rex	9	Marrón	Grande	
☐ EJP/2026/00001	Nala	4	Dorado	Pequeña	
☐ EJP/2026/00002	Simba	2	Canela	Grande	

Mascotas Mascotas					YourCompany
Nuevo Mascotas Nala :					4 / 5 < >
EJP/2026/00001					
Nombre	Nala	Color	Dorado		
Edad	4	Tamaño	Pequeña		

Reto

Cambia el prefijo para incluir el mes: `EJP/(range_year)s/(month)s/`.

Capítulo 7 · Documento con líneas y pestañas

Casi todos los documentos de Odoo (pedidos, facturas) tienen una cabecera y unas líneas. Aquí construimos ese patrón desde cero con dos modelos, pestañas y un informe PDF. Ejercicio original: `26_one2many_fields_and_notebook`.

Paso 1 · Los tres tipos de relación

Campo	Significado	Ejemplo
Many2one	Muchos registros apuntan a uno	Cada línea pertenece a un documento
One2many	El inverso: un registro tiene muchos	El documento tiene muchas líneas
Many2many	Muchos con muchos	Etiquetas de un contacto

Un `One2many` **siempre** necesita un `Many2one` en el otro modelo que apunte de vuelta.

Paso 2 · Los modelos

`models/main_model.py`:

```
from odoo import api, fields, models

class MainModel(models.Model):
    _name = 'main.model'
    _description = 'Documento'
    _order = 'id desc'

    name = fields.Char(string='Número', default='New', readonly=True, copy=False)
    company_id = fields.Many2one(
        'res.company', string='Empresa', default=lambda self: self.env.company)
    date = fields.Date(
        string='Fecha', required=True, default=fields.Date.context_today)
    user_id = fields.Many2one(
        'res.users', string='Responsable', default=lambda self: self.env.user)
    notes = fields.Text(string='Notas')
    line_ids = fields.One2many('main.model.line', 'main_model_id', string='Líneas')

    @api.model_create_multi
    def create(self, vals_list):
        for vals in vals_list:
            if vals.get('name', 'New') == 'New':
                vals['name'] = self.env['ir.sequence'].next_by_code('main.model') or 'New'
        return super().create(vals_list)

class MainModelLine(models.Model):
    _name = 'main.model.line'
    _description = 'Línea de documento'

    main_model_id = fields.Many2one(
        'main.model', string='Documento', required=True, ondelete='cascade')
    name = fields.Char(string='Nombre', required=True)
    description = fields.Char(string='Descripción')
```

`ondelete='cascade'`: al borrar el documento se borran sus líneas.

El original sobrescribía `name` en cada creación, incluso si ya traía valor; aquí solo se asigna cuando es `New`.

Paso 3 · Permisos para los dos modelos

`security/ir.access.csv`:

```
id,name,model_id,group_id/id,operation,domain
access_main_model_user,main.model usuario,main.model,base.group_user,crud,
access_main_model_line_user,main.model.line usuario,main.model.line,base.group_user,crud,
```

Paso 4 · Formulario con pestañas

```
<record id="main_model_form" model="ir.ui.view">
  <field name="name">main.model.form</field>
  <field name="model">main.model</field>
  <field name="arch" type="xml">
    <form>
      <sheet>
        <div class="oe_title">
          <h1><field name="name"/></h1>
        </div>
        <group>
          <group><field name="date"/></group>
          <group><field name="user_id"/></group>
        </group>
        <notebook>
          <page string="Líneas" name="lines">
            <field name="line_ids">
              <list editable="bottom">
                <field name="name"/>
                <field name="description"/>
              </list>
            </field>
          </page>
          <page string="Notas" name="notes">
            <field name="notes" placeholder="Escribe aquí tus notas..."/>
          </page>
        </notebook>
      </sheet>
    <chatter/>
  </form>
</field>
</record>
```

- `<notebook>` crea las pestañas y cada `<page>` es una.
- `<list editable="bottom">` permite añadir líneas directamente en la tabla, sin abrir otra ventana.
- `<chatter/>` añade mensajes y seguimiento, pero **solo** si el modelo hereda `mail.thread` (ver reto). Si no, quítalo.

Paso 5 · Lista, acción y menú

```
<record id="main_model_list" model="ir.ui.view">
  <field name="name">main.model.list</field>
  <field name="model">main.model</field>
  <field name="arch" type="xml">
    <list>
      <field name="name"/>
      <field name="date"/>
      <field name="user_id" widget="many2one_avatar_user"/>
    </list>
  </field>
</record>

<record id="action_main_model" model="ir.actions.act_window">
  <field name="name">Documentos</field>
  <field name="res_model">main.model</field>
  <field name="view_mode">list,form</field>
</record>
```

```
<menuitem id="main_model_menu_root" name="Documentos" sequence="10">
  <menuitem id="main_model_menu" name="Documentos" action="action_main_model"/>
</menuitem>
```

La secuencia `main.model` se define igual que en el capítulo 6, con prefijo `MM/`.

Paso 6 · Informe PDF básico

`report/main_model_report.xml`:

```
<odoo>
  <record id="report_main_model" model="ir.actions.report">
    <field name="name">Documento</field>
    <field name="model">main.model</field>
    <field name="report_type">qweb-pdf</field>
    <field name="report_name">ej07_one2many_notebook.main_model_document</field>
    <field name="print_report_name">'Documento - %s' % object.name</field>
    <field name="binding_model_id" ref="model_main_model"/>
    <field name="binding_type">report</field>
  </record>

  <template id="main_model_document">
    <t t-call="web.html_container">
      <t t-foreach="docs" t-as="o">
        <t t-call="web.external_layout">
          <div class="page">
            <h2>Documento <span t-field="o.name"/></h2>
            <p>Fecha: <span t-field="o.date"/></p>
            <table class="table table-sm">
              <thead>
                <tr><th>Nombre</th><th>Descripción</th></tr>
              </thead>
              <tbody>
                <tr t-foreach="o.line_ids" t-as="line">
                  <td><span t-field="line.name"/></td>
                  <td><span t-field="line.description"/></td>
                </tr>
              </tbody>
            </table>
          </div>
        </t>
      </t>
    </t>
  </template>
</odoo>
```

El original dibujaba su propia cabecera; `web.external_layout` usa la de la empresa (logo, dirección) y es lo recomendado. `t-raw` ya no existe: usa `t-out` o `t-field`.

Prueba

Crea un documento, añade tres líneas en la pestaña *Líneas* y guarda. Luego Imprimir → *Documento*.

Resultado en Odoo 20

El documento con número de secuencia, pestañas, líneas editables y el historial (`<chatter/>`, tras hacer el reto):

Documentos
Documentos
6
6
My Company (San Francisco)

Nuevo
Documentos
MM/00001
1/1
<
>

MM/00001

Fecha 30 oct Responsable OdooBot

Lineas
Notas

Nombre	Descripción	
Revisión	Revisar inventario	
Pedido	Pedir material de oficina	
Entrega	Entregar al cliente	
Añadir una línea		

Enviar mensaje
Registrar una nota

Hoy

OdooBot 12:44
Documento creado

El informe PDF (vista HTML en `/report/html/...`):



My Company (San Francisco)
8000 Marina Blvd, Suite 300
Brisbane CA 94005
Estados Unidos

Documento MM/00001

Fecha: 30/10/2026

Nombre	Descripción
Revisión	Revisar inventario
Pedido	Pedir material de oficina
Entrega	Entregar al cliente

Reto

Añade `_inherit = ['mail.thread']` al modelo y `'mail'` a `depends` para que funcione el `<chatter/>`. Después añade `tracking=True` al campo `user_id`.

Capítulo 8 · Varios teléfonos por contacto y botón inteligente

Cada contacto tendrá una lista de teléfonos, un botón con el contador en la esquina del formulario y permisos para que cada usuario vea solo los suyos. Ejercicio original: `36_partner_multi_records`.

Paso 1 · Manifiesto

```
{
    'name': 'Ej08 Teléfonos de contacto',
    'version': '20.0.1.0.0',
    'license': 'LGPL-3',
    'depends': ['contacts'],
    'data': [
        'security/security.xml',
        'security/ir.access.csv',
        'views/res_partner_phone_views.xml',
        'views/res_partner_views.xml',
    ],
}
```

Los grupos (`security.xml`) se cargan antes que `ir.access.csv`, porque el CSV los usa.

Paso 2 · Modelo de teléfonos y herencia del contacto

```
from odoo import api, fields, models
from odoo.exceptions import ValidationError

class ResPartnerPhone(models.Model):
    _name = 'res.partner.phone'
    _description = 'Teléfono de contacto'
    _order = 'id'
    _rec_name = 'number'

    partner_id = fields.Many2one(
        'res.partner', string='Contacto', required=True, ondelete='cascade')
    partner_vat = fields.Char(string='NIF', related='partner_id.vat')
    number = fields.Char(string='Número', required=True)
    notify = fields.Boolean(string='Notificaciones activas', default=True)
    user_id = fields.Many2one(
        'res.users', string='Usuario', default=lambda self: self.env.user, readonly=True)

    _number_partner_uniq = models.UniqueIndex(
        ('partner_id', 'number'), 'Este teléfono ya existe para el contacto.')

class ResPartner(models.Model):
    _inherit = 'res.partner'

    phone_ids = fields.One2many('res.partner.phone', 'partner_id', string='Teléfonos')
    phones_count = fields.Integer(compute='_compute_phones_count')

    @api.depends('phone_ids')
    def _compute_phones_count(self):
        for partner in self:
            partner.phones_count = len(partner.phone_ids)

    def action_view_partner_phones(self):
        self.ensure_one()
        action = self.env['ir.actions.act_window']._for_xml_id(
            'ej08_partner_multi_records.partner_phone_action')
        action['domain'] = [('partner_id', '=', self.id)]
```

```

        action['context'] = {'default_partner_id': self.id}
        return action

```

Novedades respecto al original:

- `models.UniqueIndex` (desde Odoo 19) sustituye al método `@api.constrains` que buscaba duplicados. La base de datos garantiza que no se repite el par contacto + número, incluso con dos usuarios a la vez.
- `_for_xml_id` sustituye a `self.env.ref(...).sudo().read()[0]` para leer una acción.
- El contador ya no se guarda (`store=True` quitado): así un usuario sin permiso sobre teléfonos puede seguir abriendo contactos.

Paso 3 · Grupos con privilegio (cambio de Odoo 19/20)

Las categorías de grupos se sustituyen por **privilegios** (`res.groups.privilege`), y el campo de usuarios de un grupo se llama ahora `user_ids`.

`security/security.xml`:

```

<odoo>
  <record id="privilege_partner_phones" model="res.groups.privilege">
    <field name="name">Teléfonos de contacto</field>
    <field name="category_id" ref="base.module_category_sales"/>
  </record>

  <record id="group_user_phones" model="res.groups">
    <field name="name">Solo mis teléfonos</field>
    <field name="privilege_id" ref="privilege_partner_phones"/>
    <field name="implied_ids" eval="[Command.link(ref('base.group_user'))]"/>
  </record>

  <record id="group_manager_phones" model="res.groups">
    <field name="name">Todos los teléfonos</field>
    <field name="privilege_id" ref="privilege_partner_phones"/>
    <field name="implied_ids" eval="[Command.link(ref('group_user_phones'))]"/>
    <field name="user_ids" eval="[Command.link(ref('base.user_admin'))]"/>
  </record>
</odoo>

```

`implied_ids` encadena grupos: quien es *Todos los teléfonos* también es *Solo mis teléfonos*.

Paso 4 · Permisos con dominio (sustituye a `ir.rule`)

En el original había un CSV de permisos más dos `ir.rule`. En Odoo 20 todo cabe en `security/ir.access.csv`:

```

id,name,model_id,group_id/id,operation,domain
access_phone_own,Teléfonos propios,res.partner.phone,ej08_partner_multi_records.group_user_phones,crud,"
[('partner_id', '=', user.partner_id.id)]"
access_phone_all,Todos los
teléfonos,res.partner.phone,ej08_partner_multi_records.group_manager_phones,crud,

```

- El grupo *Solo mis teléfonos* puede hacer de todo, pero solo con sus propios teléfonos.
- El grupo *Todos* no tiene dominio: accede a todos.
- Un usuario recibe la **suma** de los permisos de sus grupos. El responsable tiene ambos, y gana el más amplio.

Paso 5 · Vistas de teléfonos

```
<odoo>
  <record id="partner_phone_list" model="ir.ui.view">
    <field name="name">res.partner.phone.list</field>
    <field name="model">res.partner.phone</field>
    <field name="arch" type="xml">
      <list editable="bottom">
        <field name="partner_id"/>
        <field name="partner_vat"/>
        <field name="number"/>
        <field name="user_id"/>
        <field name="notify" widget="boolean_toggle"/>
      </list>
    </field>
  </record>

  <record id="partner_phone_action" model="ir.actions.act_window">
    <field name="name">Teléfonos</field>
    <field name="res_model">res.partner.phone</field>
    <field name="view_mode">list,form</field>
  </record>

  <menuitem id="partner_phone_menu" name="Teléfonos"
    parent="contacts.res_partner_menu_config"
    action="partner_phone_action"
    groups="group_user_phones" sequence="75"/>
</odoo>
```

Paso 6 · Botón inteligente en el contacto

```
<record id="view_partner_form_phones" model="ir.ui.view">
  <field name="name">res.partner.form.phones</field>
  <field name="model">res.partner</field>
  <field name="inherit_id" ref="base.view_partner_form"/>
  <field name="arch" type="xml">
    <div name="button_box" position="inside">
      <button name="action_view_partner_phones" type="object"
        class="oe_stat_button" icon="call"
        groups="ej08_partner_multi_records.group_user_phones">
        <field name="phones_count" string="Teléfonos" widget="statinfo"/>
      </button>
    </div>
  </field>
</record>
```

- `groups=` oculta el botón a quien no está en el grupo; ya no hace falta el campo `hide_btn_phones` del original.
- `widget="statinfo"` dibuja el número y la etiqueta.
- En Odoo 20 los iconos de Font Awesome (`fa-phone`) se sustituyen por Material Symbols (`call`).

Prueba

1. Como administrador, abre un contacto: aparece el botón **Teléfonos 0**. Añade dos números.
2. Intenta repetir uno: Odoo lo impide.
3. Crea un usuario de prueba con *Solo mis teléfonos* y comprueba que solo ve los teléfonos de su propio contacto.

Resultado en Odoo 20

El botón inteligente **Teléfonos** en el contacto:

The screenshot shows the Odoo 20 contact form for 'Wood Corner'. The header includes navigation links for 'Contactos', 'Contactos', and 'Configuración'. The top bar displays 'My Company (San Francisco)' with a user icon. The main content area shows the contact details for 'Wood Corner', including a logo, company name, email, phone number, address, and TIN. Below the main details, there are tabs for 'Contactos', 'Ventas y compras', 'Facturación', and 'Notas'. The 'Contactos' tab is active, showing a list of related contacts: Ron Gibson, Tom Ruiz, and Willie Burke. At the bottom, there are buttons for 'Enviar mensaje', 'Registrar una nota', and 'Actividad'.

Al pulsarlo, la lista de teléfonos de ese contacto:

The screenshot shows the Odoo 20 phone numbers list for 'Wood Corner'. The header includes navigation links for 'Contactos', 'Contactos', and 'Configuración'. The top bar displays 'My Company (San Francisco)' with a user icon. The main content area shows a table with the following columns: 'Contacto', 'NIF', 'Número', 'Usuario', and 'Notificaciones activas'. The table contains two rows of data for 'Wood Corner'.

Contacto	NIF	Número	Usuario	Notificaciones activas
☐ Wood Corner	ESB12345674	+34 600 111 222	OdooBot	☒
☐ Wood Corner	ESB12345674	+34 961 234 567	OdooBot	☐

Reto

Añade una restricción con `models.Constraint` que obligue a que `number` tenga al menos 9 caracteres: `CHECK (length(number) >= 9)`.

Parte II · Seguridad y búsqueda

Capítulo 9 · Permisos: quién puede confirmar ventas y ver precios

Crearemos un grupo *Ventas avanzado*: solo sus miembros podrán confirmar pedidos y ver precios y costes de productos. Ejercicio original: `14_sale_advance_permissions`.

Los tres niveles de seguridad

Nivel	Herramienta	Protege
Modelo y registros	<code>ir.access.csv</code>	Quién lee, crea, modifica o borra, y qué registros
Interfaz	<code>groups="..."</code> en vistas y menús	Qué ve cada usuario en pantalla
Lógica	<code>self.env.user.has_group(...)</code> en Python	Que una acción no se ejecute aunque alguien llame al método directamente

Ocultar un botón **no** es seguridad: alguien puede llamar al método por la API. Por eso este capítulo combina interfaz y lógica.

Paso 1 · Manifiesto

```
{
  'name': 'Ej09 Permisos avanzados de venta',
  'version': '20.0.1.0.0',
  'license': 'LGPL-3',
  'depends': ['sale_management'],
  'data': [
    'security/security.xml',
    'views/sale_order_views.xml',
    'views/product_views.xml',
  ],
}
```

Paso 2 · El grupo

`security/security.xml`:

```
<odoo>
  <record id="privilege_sale_advanced" model="res.groups.privilege">
    <field name="name">Ventas avanzadas</field>
    <field name="category_id" ref="base.module_category_sales"/>
  </record>

  <record id="group_sale_advanced" model="res.groups">
    <field name="name">Confirmar y ver precios</field>
    <field name="privilege_id" ref="privilege_sale_advanced"/>
    <field name="implied_ids" eval="[Command.link(ref('sales_team.group_sale_salesman'))]"/>
    <field name="user_ids" eval="[Command.link(ref('base.user_admin'))]"/>
  </record>
</odoo>
```

Cambios de Odoo 20: `users` pasa a `user_ids`, y el grupo se agrupa en la ficha del usuario mediante un **privilegio**.

Paso 3 · Ocultar los botones *Confirmar*

En vez de borrar y redibujar los botones como hacía el original (frágil si Odoo los cambia), solo les añadimos el atributo `groups`:

```
<record id="view_order_form_advanced" model="ir.ui.view">
  <field name="name">sale.order.form.advanced</field>
  <field name="model">sale.order</field>
  <field name="inherit_id" ref="sale.view_order_form"/>
  <field name="arch" type="xml">
    <xpath expr="//header/button[@id='action_confirm']" position="attributes">
      <attribute name="groups">ej09_sale_advance_permissions.group_sale_advanced</attribute>
    </xpath>
    <xpath expr="//header/button[@name='action_confirm'][not(@id)]" position="attributes">
      <attribute name="groups">ej09_sale_advance_permissions.group_sale_advanced</attribute>
    </xpath>
  </field>
</record>
```

En Odoo 20 hay dos botones *Confirmar*: uno con `id="action_confirm"` (presupuesto enviado) y otro sin `id` (borrador). La expresión `[not(@id)]` selecciona el segundo.

Paso 4 · Bloquear también en Python

`models/sale_order.py`:

```
from odoo import models
from odoo.exceptions import AccessError

class SaleOrder(models.Model):
    _inherit = 'sale.order'

    def action_confirm(self):
        if not self.env.user.has_group('ej09_sale_advance_permissions.group_sale_advanced'):
            raise AccessError(self.env._('Solo el grupo Ventas avanzadas puede confirmar pedidos.'))
        return super().action_confirm()
```

Recuerda añadir `models/__init__.py` y el `__init__.py` raíz. El original importaba `xlrd`, `csv` y `tempfile` sin usarlos: sobran.

Paso 5 · Ocultar precios y costes

`views/product_views.xml`:

```
<odoo>
  <!-- Lista de plantillas de producto -->
  <record id="product_template_list_advanced" model="ir.ui.view">
    <field name="name">product.template.list.advanced</field>
    <field name="model">product.template</field>
    <field name="inherit_id" ref="product.product_template_tree_view"/>
    <field name="arch" type="xml">
      <field name="list_price" position="attributes">
        <attribute name="groups">ej09_sale_advance_permissions.group_sale_advanced</attribute>
      </field>
      <field name="standard_price" position="attributes">
        <attribute name="groups">ej09_sale_advance_permissions.group_sale_advanced</attribute>
      </field>
    </field>
  </record>

  <!-- Formulario de producto -->
```

```

<record id="product_template_form_advanced" model="ir.ui.view">
  <field name="name">product.template.form.advanced</field>
  <field name="model">product.template</field>
  <field name="inherit_id" ref="product.product_template_form_view"/>
  <field name="arch" type="xml">
    <field name="standard_price" position="attributes">
      <attribute name="groups">ej09_sale_advance_permissions.group_sale_advanced</attribute>
    </field>
  </field>
</record>
</odoo>

```

El id de la lista sigue siendo `product_template_tree_view` aunque la etiqueta sea `<list>`: Odoo no renombra los XML ID para no romper módulos.

Paso 6 · Asigna el grupo

Ajustes → Usuarios → abre un usuario. En la sección *Ventas* verás el privilegio *Ventas avanzadas*; marca *Confirmar* y *ver precios*.

Prueba

1. Con un vendedor sin el grupo: crea un presupuesto. No aparece *Confirmar* y la lista de productos no muestra precios.
2. Con el administrador: todo es visible y se puede confirmar.

Resultado en Odoo 20

Con el usuario *Marc Demo* (vendedor sin el grupo avanzado), el presupuesto no tiene botón *Confirmar*:

The screenshot displays the Odoo 20 interface for a sales quotation. The top navigation bar includes 'Ventas', 'Pedidos', and 'Productos'. The user is logged in as 'My Company (San Francisco)'. The main content area shows a quotation titled 'S00005' for 'Acme Corporation'. The form includes fields for 'Cliente', 'Vencimiento', 'Condiciones de pago', and 'Fecha de entrega'. Below these, there is a table of line items with columns for 'Descripción', 'Cantidad', 'Precio unita...', 'Impuestos', and 'Importe'. The total amount is \$465.75, and the margin is 15.00%.

Descripción	Cantidad	Precio unita...	Impuestos	Importe
[FURN_8888] Lámpara de oficina	1,00	405,00	15%	\$ 405,00

Términos y condiciones...
 Importe base: \$ 405,00
 Tax 15%: \$ 60,75
 Total: \$ 465,75
 Margin (4%): \$ 15,00

Y la lista de productos no muestra precio de venta ni coste:

<div> <div>Ventas Pedidos Productos</div> <div> <div> <div>9</div> <div>3</div> </div> <div>My Company (San Francisco)</div> </div> </div>				
<div> <div>Productos</div> <div> <div> <div>Q</div> <div>Ventas</div> <div>×</div> <div>Buscar...</div> </div> <div>1-38 / 38</div> <div> <div><</div> <div>></div> <div></div> <div></div> <div></div> </div> </div> </div>				
☐	Nombre del producto	Referencia interna	Impuesto de ventas	Stock real
☐	☆ AGUA	P002	15%	0,00
☐	☆ Alfombrilla de escritorio	FURN_0002	15%	0,00
☐	☆ Alojamiento de hotel		15%	
☐	☆ Armario con puertas	E-COM11	15%	33,00
☐	☆ Caja de administración de cables	FURN_5555	15%	90,00
☐	☆ Caja de almacenamiento	E-COM08	15%	18,00
☐	☆ Cajón	FURN_8855	15%	80,00
☐	☆ Cajón negro	FURN_8900	15%	0,00
☐	☆ Combinación de escritorio	FURN_7800	15%	2.855,00
☐	☆ Cubo de basura con pedal	E-COM10	15%	22,00
☐	☆ Depósito			
☐	☆ Diseño interior virtual		15%	
☐	☆ Escritorio en L con asiento a la derecha	E-COM06	15%	0,00
☐	☆ Escritorio en L con asiento a la izquierda	FURN_1118	15%	2,00
☐	☆ Escritorio grande	E-COM09	15%	0,00
☐	☆ Escritorio para cuatro personas	FURN_8220	15%	20,00
☐	☆ Escritorio personalizable		15%	285,00

Reto

Permite que cualquier vendedor confirme pedidos de menos de 500 €, y exige el grupo solo por encima de esa cantidad (`self.amount_total`).

Capítulo 10 · Filtros y agrupaciones personalizadas

Añadiremos al buscador de Contactos dos filtros (*Sin NIF* y *Con NIF*), dos campos de búsqueda (ciudad y código postal) y dos agrupaciones. Ejercicio original: `15_partner_custom_filter`.

Paso 1 · Qué es un dominio

Un dominio es una lista de condiciones con la forma `(campo, operador, valor)`:

```
[('vat', '=', False)]          # sin NIF
[('vat', '!=', False)]         # con NIF
[('city', 'ilike', 'valencia')] # ciudad contiene "valencia", sin distinguir mayúsculas
['|', ('city', '=', 'Valencia'), ('city', '=', 'Madrid')] # 0 lógico
```

Varias condiciones seguidas se combinan con `Y`. Para `O` se antepone `'|'`. Operadores habituales: `=`, `!=`, `>`, `<`, `in`, `not in`, `ilike`, `child_of`.

En Python, desde Odoo 19, puedes construirlos de forma más segura con la clase `Domain`:

```
from odoo.fields import Domain

dominio = Domain('vat', '=', False) & Domain('city', 'ilike', 'valencia')
partners = self.env['res.partner'].search(dominio)
```

Paso 2 · Hereda la vista de búsqueda

```
<?xml version="1.0" encoding="utf-8"?>
<odoo>
  <record id="res_partner_search_custom" model="ir.ui.view">
    <field name="name">res.partner.search.custom</field>
    <field name="model">res.partner</field>
    <field name="inherit_id" ref="base.view_res_partner_filter"/>
    <field name="arch" type="xml">
      <!-- Filtros: antes de "Archivado" -->
      <xpath expr="//filter[@name='inactive']" position="before">
        <filter string="Sin NIF" name="no_vat" domain="[('vat', '=', False)]"/>
        <filter string="Con NIF" name="with_vat" domain="[('vat', '!=', False)]"/>
        <separator/>
      </xpath>

      <!-- Campos donde se puede escribir para buscar -->
      <field name="user_id" position="after">
        <field name="city"/>
        <field name="zip" string="Código postal"/>
      </field>

      <!-- Agrupar por -->
      <xpath expr="//group" position="inside">
        <filter string="Ciudad" name="group_by_city" context="{ 'group_by': 'city' }"/>
        <filter string="Código postal" name="group_by_zip" context="{ 'group_by': 'zip' }"/>
      </xpath>
    </field>
  </record>
</odoo>
```

Elemento	Para qué sirve
<code><filter domain=...></code>	Botón en el panel <i>Filtros</i>
<code><separator/></code>	Filtros separados se combinan con <code>Y</code> ; dentro del mismo bloque, con <code>O</code>
<code><field></code>	Al escribir en el buscador, ofrece "Buscar Ciudad por..."

Elemento	Para qué sirve
<code><filter context="{ 'group_by': ... }"></code>	Opción en <i>Agrupar por</i>

Sin NIF y *Con NIF* están en el mismo bloque: si marcas los dos se combinan con O y verás todos los contactos.

Paso 3 · Manifiesto

```
'depends': ['contacts'],
'data': ['views/res_partner_views.xml'],
```

Paso 4 · Un filtro activo por defecto

Para que la lista de contactos se abra ya filtrada, crea una acción propia con contexto:

```
<record id="action_partner_no_vat" model="ir.actions.act_window">
  <field name="name">Contactos sin NIF</field>
  <field name="res_model">res.partner</field>
  <field name="view_mode">list,form</field>
  <field name="context">{'search_default_no_vat': 1}</field>
</record>
```

`search_default_<nombre del filtro>` activa ese filtro al abrir.

Filtros con fechas

```
<filter string="Creados este mes" name="this_month"
  domain="[('create_date', '>=', (context_today() - relativedelta(day=1)).strftime('%Y-%m-%d'))]" />
<filter string="Fecha de creación" name="create_date" date="create_date" />
```

El segundo, con `date=`, crea automáticamente el desplegable de meses, trimestres y años. En XML, `>` se escribe `>=`.

Prueba

Contactos — Filtros — *Sin NIF*. Después Agrupar por — *Ciudad*. Escribe "Valencia" en el buscador y elige *Buscar Ciudad por: Valencia*.

Resultado en Odoo 20

Los filtros *Sin NIF* y *Con NIF* y las agrupaciones *Ciudad* y *Código postal* en el buscador de contactos:

Contactos			My Company (San Francisco)		
Nuevo Contactos			1-63 / 63		
<div> <div>Nombre</div> <div> ☐ Abigail Peterson ☐ Acme Corporation ☐ Acme Corporation, Addison Olson ☐ Acme Corporation, Douglas Fletcher ☐ Acme Corporation, Floyd Steward ☐ Anita Oliver ☐ Audrey Peterson ☐ Azure Interior ☐ Azure Interior, Brandon Freeman </div> </div>			<div> <div>Filtros</div> <div> Subject to VAT Facturas de cliente Facturas de proveed... Facturas vencidas Empleados Sin NIF Con NIF Archivado Filtro personalizado... </div> </div>		
			<div> <div>Agrupar por</div> <div> Comercial Compañía País Propiedades Ciudad Código postal Grupo personalizado </div> </div>		
			<div> <div>Favoritos</div> <div> Save search 7 2 3 </div> </div>		
☐ Azure Interior, Colleen Diaz			colleen.diaz83@example.com +91 0000 123 7... Estados U...		
☐ Azure Interior, Nicole Ford			nicole.ford75@example.com +91 0000 123 7... Estados U...		
☐ Barty McBly			barty.mcbly@example.com (555)-234-3424		
☐ Beth Evans			beth.evans77@example.com (555)-532-3841		
☐ Doris Cole			doris.cole31@example.com (555)-331-5378		
☐ Eli Lambert			eli.lambert22@example.com (555)-169-1352		
☐ Ernest Reed			ernest.reed47@example.com (555)-518-8232		
☐ Gemini Furniture			gemini_furniture@fake.geminifurniture.com +378 0549 885... San Marino 2 17		

Reto

Añade un filtro *Empresas de España* con `[('is_company', '=', True), ('country_id.code', '=', 'ES')]`.

Parte III · Informes

Capítulo 11 · Informes PDF: personalizar la factura

Los PDF de Odoo se escriben en **QWeb**, una plantilla HTML con instrucciones `t-`. Hay dos estrategias: **heredar** la factura estándar y cambiar solo una parte, o **crear** un informe nuevo desde cero. Ejercicio original: `07_modify_invoice_report`.

El original sustituía la factura entera con la etiqueta `<report>`, que ya no existe en Odoo 20. Aquí verás las dos formas actuales.

QWeb en 2 minutos

Instrucción	Qué hace
<code>t-foreach="docs" t-as="o"</code>	Recorre los registros que se imprimen
<code>t-field="o.amount_total"</code>	Muestra un campo con su formato (moneda, fecha)
<code>t-out="expresión"</code>	Muestra el resultado de una expresión (sustituye a <code>t-esc</code> y <code>t-raw</code>)
<code>t-if="condición"</code>	Muestra el bloque solo si se cumple
<code>t-set="x" t-value="..."</code>	Crea una variable
<code>t-call="plantilla"</code>	Inserta otra plantilla

Opción A · Heredar la factura estándar (recomendada)

La plantilla del cuerpo de la factura es `account.report_invoice_document`. Añadimos una línea bajo el bloque de fechas y un texto legal al final.

`report/invoice_report.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<odoo>
  <template id="report_invoice_document_custom"
    inherit_id="account.report_invoice_document">
    <!-- Debajo de fechas y referencias -->
    <xpath expr="//div[@id='informations']" position="after">
      <p class="mb-3">
        <strong>NIF cliente:</strong>
        <span t-field="o.partner_id.vat"/>
      </p>
    </xpath>
    <!-- Al final, después de las notas -->
    <xpath expr="//div[@name='comment']" position="after">
      <p class="text-muted small">
        Gracias por su confianza. Factura emitida por <span t-field="o.company_id.name"/>.
      </p>
    </xpath>
  </template>
</odoo>
```

Manifiesto: `'depends': ['account']` y `'data': ['report/invoice_report.xml']`.

Cómo encontrar dónde engancharte: activa el modo desarrollador, ve a Ajustes → Técnico → Vistas, busca `report_invoice_document` y localiza elementos con `name=` o `id=`. Son los anclajes estables.

Opción B · Un informe nuevo junto al estándar

En vez de reemplazar la factura oficial (que en España debe cumplir requisitos legales), crea un segundo PDF, por ejemplo una *factura resumida*:

```
<odoo>
<record id="report_invoice_summary" model="ir.actions.report">
  <field name="name">Factura resumida</field>
  <field name="model">account.move</field>
  <field name="report_type">qweb-pdf</field>
  <field name="report_name">ejll_invoice_report.invoice_summary</field>
  <field name="print_report_name">'Resumen - %s' % (object.name or 'borrador')</field>
  <field name="binding_model_id" ref="account.model_account_move"/>
  <field name="binding_type">report</field>
  <field name="paperformat_id" ref="paperformat_invoice_summary"/>
</record>

<template id="invoice_summary">
  <t t-call="web.html_container">
    <t t-foreach="docs" t-as="o">
      <t t-call="web.external_layout">
        <div class="page">
          <h2>Factura <span t-field="o.name"/></h2>
          <p>
            Cliente: <span t-field="o.partner_id"/> ·
            Fecha: <span t-field="o.invoice_date"/>
          </p>
          <table class="table table-sm">
            <thead>
              <tr>
                <th>Artículo</th>
                <th class="text-end">Cantidad</th>
                <th class="text-end">Precio</th>
                <th class="text-end">Desc. %</th>
                <th class="text-end">Importe</th>
              </tr>
            </thead>
            <tbody>
              <t t-foreach="o.invoice_line_ids.filtered(lambda l: l.display_type ==
'product')" t-as="line">
                <tr>
                  <td><span t-field="line.name"/></td>
                  <td class="text-end"><span t-field="line.quantity"/></td>
                  <td class="text-end"><span t-field="line.price_unit"/></td>
                  <td class="text-end"><span t-field="line.discount"/></td>
                  <td class="text-end"><span t-field="line.price_subtotal"/></td>
                </tr>
              </t>
            </tbody>
          </table>
          <div class="text-end">
            <p>Base imponible: <span t-field="o.amount_untaxed"/></p>
            <p>IVA: <span t-field="o.amount_tax"/></p>
            <h4>Total: <span t-field="o.amount_total"/></h4>
          </div>
        </div>
      </t>
    </t>
  </t>
</template>
</odoo>
```

- `binding_model_id` + `binding_type=report` hacen que aparezca en el menú **Imprimir** de la factura.
- `display_type == 'product'` excluye las líneas de sección y nota.

- `web.external_layout` añade la cabecera y el pie de la empresa. Bootstrap 5: `text-end` en lugar del antiguo `text-right`.

Formato de papel

El original definía un `report.paperformat`. Sigue funcionando igual; decláralo **antes** del informe que lo usa:

```
<record id="paperformat_invoice_summary" model="report.paperformat">
  <field name="name">A4 factura resumida</field>
  <field name="format">A4</field>
  <field name="orientation">Portrait</field>
  <field name="margin_top">40</field>
  <field name="margin_bottom">25</field>
  <field name="margin_left">7</field>
  <field name="margin_right">7</field>
  <field name="header_spacing">35</field>
  <field name="dpi">90</field>
</record>
```

Ver el informe en HTML mientras lo diseñas

Para no generar un PDF cada vez, abre en el navegador:

```
http://localhost:8069/report/html/ej11_invoice_report.invoice_summary/ID
```

cambiando `ID` por el id de una factura. Con `--dev=xml` los cambios en la plantilla se ven al recargar.

Prueba

Abre una factura confirmada — Imprimir. Verás *Factura PDF* con el NIF y el texto legal añadidos, y *Factura resumida*.

Resultado en Odoo 20

Opción A: la factura estándar con la línea *NIF cliente* y el texto final añadidos:



My Company (San Francisco)
8000 Marina Blvd, Suite 300
Brisbane CA 94005
Estados Unidos

Acme Corporation
77 Santa Barbara Rd
Pleasant Hill CA 94523
Estados Unidos

TIN: US12345673

Factura INV/2026/00010

Fecha de factura

30/10/2026

Fecha de vencimiento

29/11/2026

NIF cliente: US12345673

Descripción	Cantidad	Precio unitario	Impuestos	Importe
[DESK0005] Escritorio personalizable (Blanco, Personalizado) 160 x 80 cm, con patas grandes	2,00	750,00	15%	\$ 1.500,00
Condiciones de pago: 30 días				
Comunicación de pago: INV/2026/00010 en esta cuenta: BANK134567890				
Gracias por su confianza. Factura emitida por My Company (San Francisco).				
				Importe base \$ 1.500,00
				Tax 15% \$ 225,00
				Total \$ 1.725,00
				Pagado el 30/10/2026 \$ 1.725,00
				Importe adeudado \$ 0,00

Opción B: la factura resumida:



My Company (San Francisco)
8000 Marina Blvd, Suite 300
Brisbane CA 94005
Estados Unidos

Factura INV/2026/00010

Cliente: Acme Corporation · Fecha: 30/10/2026

Artículo	Cantidad	Precio	Desc. %	Importe
160 x 80 cm, con patas grandes	2,00	750,00	0,00	\$ 1.500,00
				Base imponible: \$ 1.500,00
				IVA: \$ 225,00
				Total: \$ 1.725,00

Reto

En la factura resumida, muestra la columna Desc. % solo si alguna línea tiene descuento (`t-if="any(l.discount for l in o.invoice_line_ids)"`).

Capítulo 12 · Modificar el presupuesto de venta

El ejercicio más corto del libro: añadir un bloque al PDF de presupuestos y pedidos con una sola herencia. Ejercicio original: `08_modify_sale_report`.

Paso 1 · Manifiesto

```
{
  'name': 'Ej12 Informe de venta modificado',
  'version': '20.0.1.0.0',
  'license': 'LGPL-3',
  'depends': ['sale'],
  'data': ['report/sale_report.xml'],
}
```

Paso 2 · La herencia

La plantilla del cuerpo del pedido sigue llamándose `sale.report_saleorder_document` en Odoo 20, y el bloque de firma conserva `name="signature"`:

```
<?xml version="1.0" encoding="utf-8"?>
<odoo>
  <template id="report_saleorder_document_custom"
    inherit_id="sale.report_saleorder_document">
    <xpath expr="//div[@name='signature']" position="after">
      <div class="mt-4 p-3 border rounded">
        <h5>Condiciones</h5>
        <p class="mb-1">Validez del presupuesto: 30 días.</p>
        <p class="mb-0" t-if="doc.user_id">
          Su comercial: <span t-field="doc.user_id"/>
          (<span t-field="doc.user_id.email"/>)
        </p>
      </div>
    </xpath>
  </template>
</odoo>
```

Fíjate en la variable: en el informe de venta el registro se llama `doc`, mientras que en la factura se llama `o`. Cada informe elige su nombre; míralo en la plantilla original antes de escribir.

En lugar del `style="color: red;"` del original usamos clases de Bootstrap 5 (`border`, `rounded`, `mt-4`), que respetan el diseño de la empresa.

Paso 3 · Mostrar algo solo en presupuestos

```
<p t-if="doc.state in ('draft', 'sent')">Este documento es un presupuesto, no un pedido confirmado.</p>
```

Paso 4 · Añadir una columna a las líneas

La tabla de líneas tiene la clase `o_main_table`. Para añadir la referencia interna del producto:

```
<xpath expr="//table[hasclass('o_main_table')]/thead/tr/th[1]" position="before">
  <th class="text-start">Ref.</th>
</xpath>
<xpath expr="//table[hasclass('o_main_table')]/tbody//tr[1]/td[1]" position="before">
```

```
<td><span t-field="line.product_id.default_code" /></td>
</xpath>
```

Esta es la parte delicada: la tabla de ventas tiene filas de sección y nota además de las de producto. Si la columna descuadra la tabla, revisa la plantilla original y engancha la celda en la fila correcta.

Prueba

Ventas — abre un presupuesto — Imprimir — *Presupuesto / Pedido*. El bloque *Condiciones* aparece después de la firma.

Resultado en Odoo 20

El bloque *Condiciones* al final del pedido:



My Company (San Francisco)
8000 Marina Blvd, Suite 300
Brisbane CA 94005
Estados Unidos

Invoicing and Shipping Address
Gemini Furniture, Oscar Morgan
Via Industria 21
Serravalle 47899
San Marino
☎ +32 18 123 460

Gemini Furniture
Via Industria 21
Serravalle 47899
San Marino
TIN: SM12345

Pedido nº S00024

Fecha del pedido
30/10/2026

Contacto
Marc Demo

	Precio unitario	Impuestos	Importe
[E-COM06] Escritorio en L con asiento a la derecha	10,00	199,00 0% Exports	\$ 1.990,00
Escritorio alineado a la derecha con espacio de almacenamiento	Unidades		
		Importe base	\$ 1.990,00
		Tax 0%	\$ 0,00
		Total	\$ 1.990,00

Condiciones

Validez del presupuesto: 30 días.
Su comercial: Marc Demo (mark.brown23@example.com)

Reto

Muestra en el bloque el total de unidades del pedido: `sum(doc.order_line.mapped('product_uom_qty'))`.

Parte IV · Asistentes y configuración

Capítulo 13 · Asistente desde el menú Acción

Un asistente (wizard) es una ventana emergente que pide unos datos y ejecuta algo. Aquí seleccionaremos varios productos en la lista y les cambiaremos los impuestos de venta y compra de una vez. Ejercicio original:

11_wizard_from_action.

Paso 1 · `TransientModel`

Un asistente usa `models.TransientModel`: sus registros son temporales y Odoo los borra solo.

wizard/modify_product.py:

```
from odoo import fields, models
from odoo.exceptions import UserError

class ModifyProductWizard(models.TransientModel):
    _name = 'modify.product.wzd'
    _description = 'Cambiar impuestos de productos'

    company_id = fields.Many2one(
        'res.company', string='Empresa', default=lambda self: self.env.company, readonly=True)
    sale_tax_ids = fields.Many2many(
        'account.tax', 'modify_product_wzd_sale_tax_rel', string='Impuestos de venta',
        domain="['type_tax_use', '=', 'sale'], ('company_id', '=', company_id)")
    purchase_tax_ids = fields.Many2many(
        'account.tax', 'modify_product_wzd_purchase_tax_rel', string='Impuestos de compra',
        domain="['type_tax_use', '=', 'purchase'], ('company_id', '=', company_id)")

    def action_modify_products(self):
        self.ensure_one()
        if not self.sale_tax_ids and not self.purchase_tax_ids:
            raise UserError(self.env._('Selecciona al menos un impuesto.'))
        templates = self.env['product.template'].browse(self.env.context.get('active_ids', []))
        vals = {}
        if self.sale_tax_ids:
            vals['taxes_id'] = [fields.Command.set(self.sale_tax_ids.ids)]
        if self.purchase_tax_ids:
            vals['supplier_taxes_id'] = [fields.Command.set(self.purchase_tax_ids.ids)]
        templates.write(vals)
        return {'type': 'ir.actions.act_window_close'}
```

Claves:

- `active_ids` en el contexto contiene los ids de los registros seleccionados en la lista.
- Un solo `write` sobre todos los productos es mucho más rápido que un bucle con asignaciones.
- `Command.set(ids)` sustituye todo el `Many2many` por esa lista. Otras órdenes: `Command.link(id)` añade, `Command.unlink(id)` quita, `Command.clear()` vacía.
- Usamos `Many2many` porque un producto puede tener varios impuestos. Dos `Many2many` al mismo modelo necesitan nombres de tabla distintos (el segundo argumento).

Paso 2 · Permiso

Los `TransientModel` también necesitan permiso en Odoo 20:

```
id,name,model_id/group_id/id,operation,domain
access_modify_product_wzd,modify.product.wzd,modify.product.wzd,base.group_user,crud,
```

Paso 3 · Vista y acción enlazada

```
<?xml version="1.0" encoding="utf-8"?>
<odoo>
  <record id="modify_product_wzd_form" model="ir.ui.view">
    <field name="name">modify.product.wzd.form</field>
    <field name="model">modify.product.wzd</field>
    <field name="arch" type="xml">
      <form>
        <group>
          <group>
            <field name="sale_tax_ids" widget="many2many_tags"
              options="{ 'no_create': True }"/>
          </group>
          <group>
            <field name="purchase_tax_ids" widget="many2many_tags"
              options="{ 'no_create': True }"/>
            <field name="company_id" invisible="1"/>
          </group>
        </group>
        <footer>
          <button string="Aplicar" name="action_modify_products"
            type="object" class="btn-primary"/>
          <button string="Cancelar" special="cancel" class="btn-secondary"/>
        </footer>
      </form>
    </field>
  </record>

  <record id="action_modify_product" model="ir.actions.act_window">
    <field name="name">Cambiar impuestos</field>
    <field name="res_model">modify.product.wzd</field>
    <field name="view_mode">form</field>
    <field name="target">new</field>
    <field name="binding_model_id" ref="product.model_product_template"/>
    <field name="binding_view_types">list,form</field>
  </record>
</odoo>
```

Campo de la acción	Efecto
<code>target = new</code>	Se abre como ventana emergente
<code>binding_model_id</code>	Aparece en el menú Acción (engranaje) de ese modelo
<code>binding_view_types</code>	En qué vistas aparece: lista, formulario o ambas
<code><footer></code>	Botones al pie del diálogo; <code>special="cancel"</code> cierra sin hacer nada

Paso 4 · Manifiesto

```
'depends': ['account'],
'data': [
  'security/ir.access.csv',
  'wizard/modify_product_views.xml',
],
```

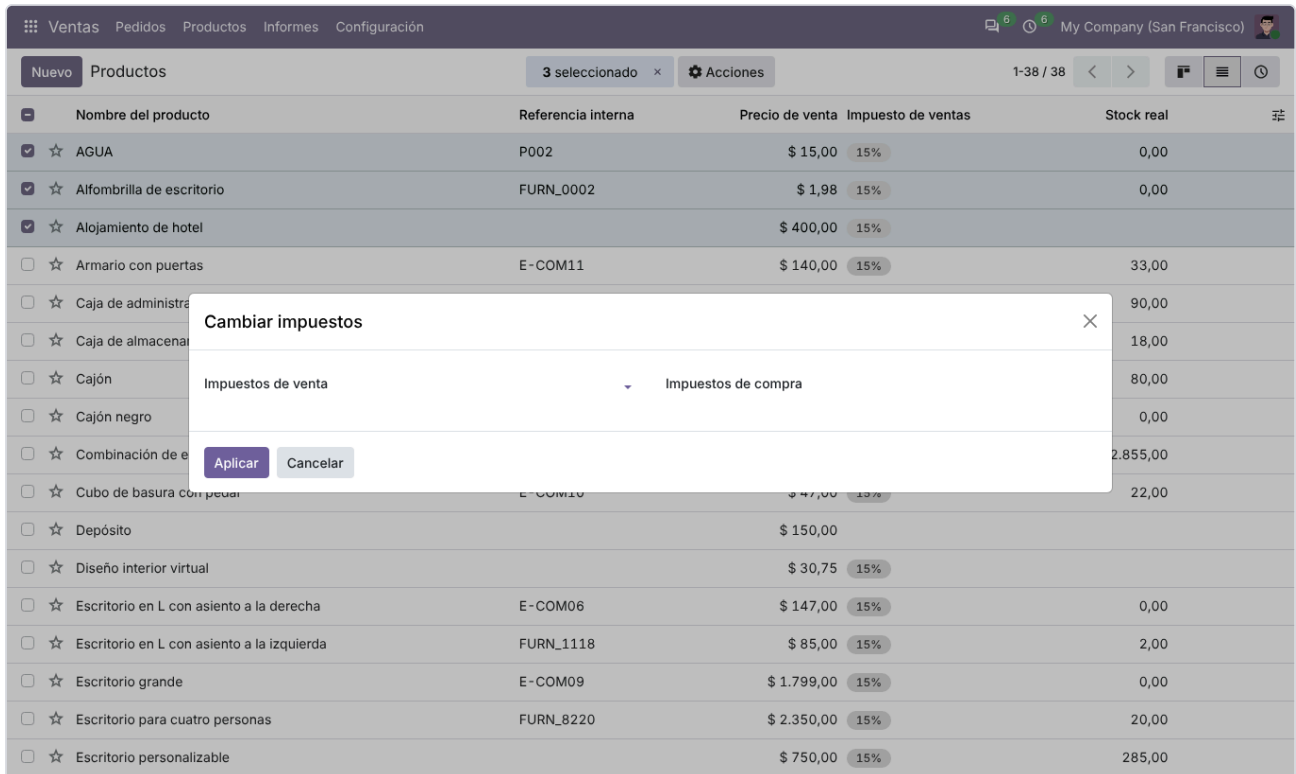
Los ficheros Python del asistente van en `wizard/` y hay que importar esa carpeta en el `__init__.py` raíz: `from . import wizard`.

Prueba

Ventas → Productos → vista lista. Marca tres productos → Acción (engranaje) → *Cambiar impuestos*. Elige IVA 21 % en venta y aplica. Abre un producto y comprueba la pestaña *Contabilidad*.

Resultado en Odoo 20

Tres productos seleccionados → Acciones → *Cambiar impuestos*:



	Nombre del producto	Referencia interna	Precio de venta	Impuesto de ventas	Stock real
☒	☆ AGUA	P002	\$ 15,00	15%	0,00
☒	☆ Alfombrilla de escritorio	FURN_0002	\$ 1,98	15%	0,00
☒	☆ Alojamiento de hotel		\$ 400,00	15%	
☐	☆ Armario con puertas	E-COM11	\$ 140,00	15%	33,00
☐	☆ Caja de administra				90,00
☐	☆ Caja de almacenar				18,00
☐	☆ Cajón				80,00
☐	☆ Cajón negro				0,00
☐	☆ Combinación de e				2.855,00
☐	☆ Cubo de basura con pedal	E-COM10	\$ 47,00	15%	22,00
☐	☆ Depósito		\$ 150,00		
☐	☆ Diseño interior virtual		\$ 30,75	15%	
☐	☆ Escritorio en L con asiento a la derecha	E-COM06	\$ 147,00	15%	0,00
☐	☆ Escritorio en L con asiento a la izquierda	FURN_1118	\$ 85,00	15%	2,00
☐	☆ Escritorio grande	E-COM09	\$ 1.799,00	15%	0,00
☐	☆ Escritorio para cuatro personas	FURN_8220	\$ 2.350,00	15%	20,00
☐	☆ Escritorio personalizable		\$ 750,00	15%	285,00

Reto

Añade al asistente un campo `list_price` opcional que, si tiene valor, cambie también el precio de venta de los productos seleccionados.

Capítulo 14 · Fechas: buscar ventas por rango

Un asistente pedirá un rango de fechas y abrirá la lista de pedidos de venta de ese periodo. Aprenderás a operar con fechas y a devolver una acción desde Python. Ejercicio original: `23_search_sale_by_date`.

Paso 1 · Date y Datetime

Tipo de campo	Valor en Python	Ejemplo	Ojo
<code>fields.Date</code>	<code>datetime.date</code>	<code>date_order.date()</code>	Sin hora ni zona horaria
<code>fields.Datetime</code>	<code>datetime.datetime</code>	<code>sale.date_order</code>	Se guarda en UTC; la pantalla lo muestra en la hora del usuario

Ayudas útiles:

```
from datetime import timedelta
from dateutil.relativedelta import relativedelta
from odoo import fields

hoy = fields.Date.context_today(self)      # fecha de hoy según la zona del usuario
ahora = fields.Datetime.now()              # ahora, en UTC
hace_7 = hoy - timedelta(days=7)
inicio_mes = hoy + relativedelta(day=1)
fin_mes = hoy + relativedelta(day=31)      # relativedelta ajusta al último día real
```

Paso 2 · El asistente

```
from datetime import datetime, time, timedelta

from odoo import api, fields, models
from odoo.exceptions import UserError, ValidationError

class SearchSaleByDate(models.TransientModel):
    _name = 'search.sale.by.date'
    _description = 'Buscar ventas por fecha'

    company_id = fields.Many2one(
        'res.company', string='Empresa', default=lambda self: self.env.company)
    date_from = fields.Date(string='Desde', required=True,
        default=lambda self: fields.Date.context_today(self) - timedelta(days=30))
    date_to = fields.Date(string='Hasta', required=True, default=fields.Date.context_today)
    days_ago = fields.Integer(string='0 los últimos N días')

    @api.onchange('days_ago')
    def _onchange_days_ago(self):
        if self.days_ago > 0:
            self.date_to = fields.Date.context_today(self)
            self.date_from = self.date_to - timedelta(days=self.days_ago)

    @api.constrains('date_from', 'date_to')
    def _check_dates(self):
        for wizard in self:
            if wizard.date_from > wizard.date_to:
                raise ValidationError(self.env._('La fecha inicial debe ser anterior a la final.'))

    def action_search_sales(self):
        self.ensure_one()
        # date_order es Datetime: cubrimos el día final completo
        start = datetime.combine(self.date_from, time.min)
        end = datetime.combine(self.date_to, time.max)
        orders = self.env['sale.order'].search([
            ('date_order', '>=', start),
```

```

        ('date_order', '<=', end),
        ('company_id', '=', self.company_id.id),
    ])
    if not orders:
        raise UserError(self.env._('No hay ventas en ese periodo.'))
    action = self.env['ir.actions.act_window']._for_xml_id('sale.action_orders')
    action['domain'] = [('id', 'in', orders.ids)]
    action['context'] = {'create': False}
    return action

```

Mejoras sobre el original:

1. El original buscaba con `date_order <= fecha`, es decir, **todas** las ventas anteriores. Ahora hay un rango real.
2. `@api.onchange` rellena las fechas al escribir los días, sin guardar.
3. `@api.constrains` impide un rango al revés.
4. `datetime.combine(..., time.max)` incluye las ventas de todo el último día.
5. `_for_xml_id` sustituye a `env.ref(...).read()[0]`.

Paso 3 · Vista, acción y menú

```

<odoo>
<record id="search_sale_by_date_form" model="ir.ui.view">
  <field name="name">search.sale.by.date.form</field>
  <field name="model">search.sale.by.date</field>
  <field name="arch" type="xml">
    <form>
      <group>
        <group>
          <field name="date_from"/>
          <field name="date_to"/>
        </group>
        <group>
          <field name="days_ago"/>
        </group>
      </group>
      <footer>
        <button string="Buscar" name="action_search_sales" type="object" class="btn-primary"/>
        <button string="Cancelar" special="cancel" class="btn-secondary"/>
      </footer>
    </form>
  </field>
</record>

<record id="action_search_sale_by_date" model="ir.actions.act_window">
  <field name="name">Buscar ventas por fecha</field>
  <field name="res_model">search.sale.by.date</field>
  <field name="view_mode">form</field>
  <field name="target">new</field>
</record>

<menuitem id="menu_search_sale_by_date" name="Buscar por fecha"
  parent="sale.sale_order_menu" action="action_search_sale_by_date" sequence="90"/>
</odoo>

```

Permiso en `security/ir.access.csv`:

```

id,name,model_id,group_id/id,operation,domain
access_search_sale_by_date,search.sale.by.date,search.sale.by.date,sales_team.group_sale_salesman,crud,

```

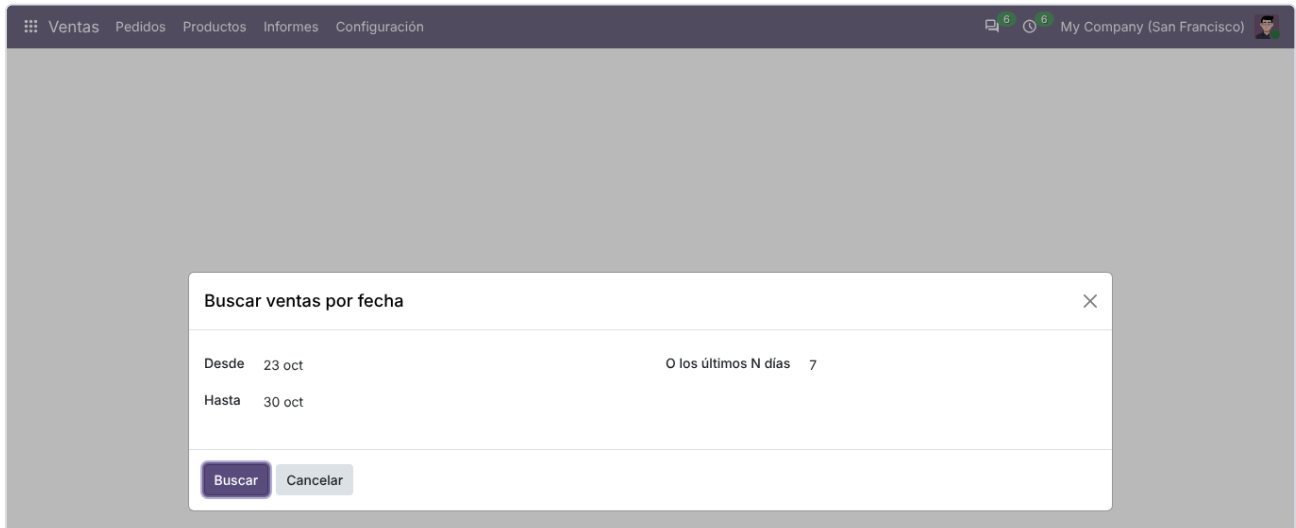
Manifiesto: `'depends': ['sale']`.

Prueba

Ventas — Pedidos — *Buscar por fecha*. Escribe 7 en *O los últimos N días*: las fechas cambian solas. Pulsa *Buscar* y se abre la lista filtrada. Prueba un rango al revés para ver el error.

Resultado en Odoo 20

Al escribir 7 días, el `onchange` rellena las fechas:



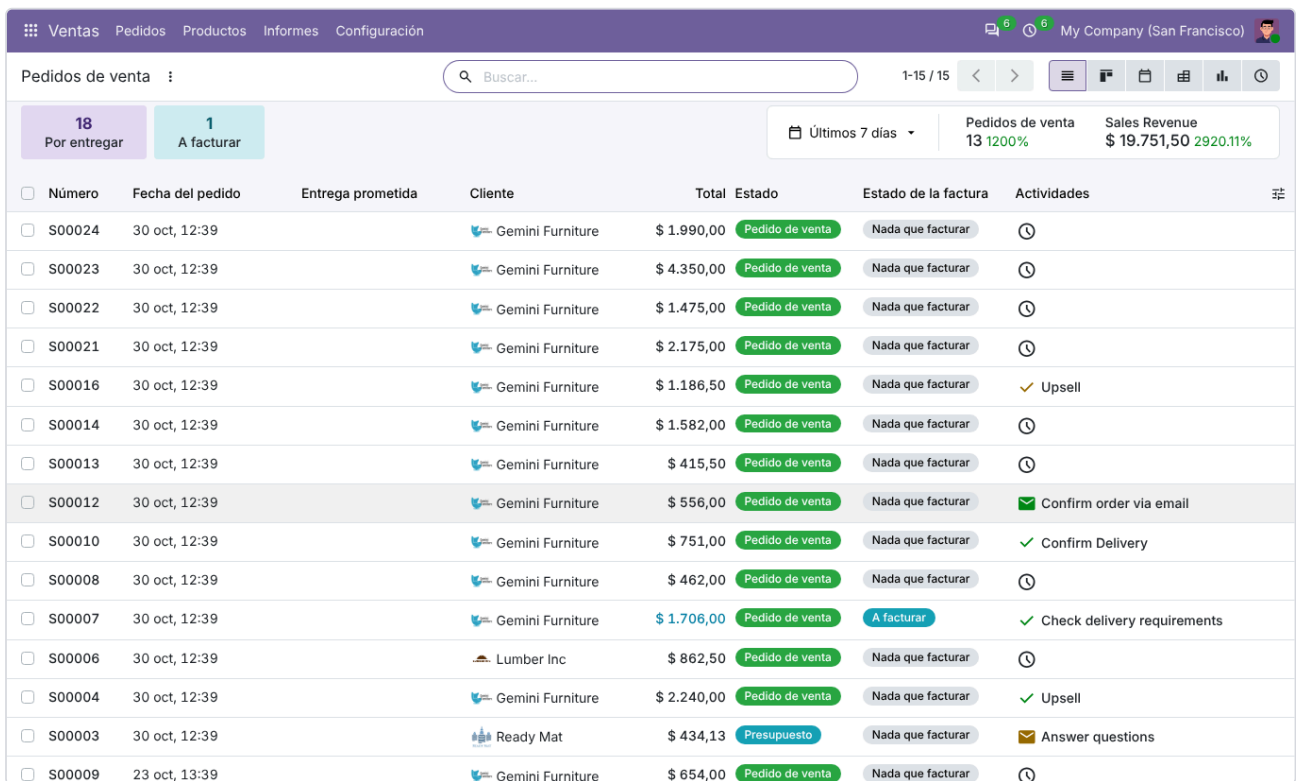
Buscar ventas por fecha

Desde 23 oct O los últimos N días 7

Hasta 30 oct

Buscar Cancelar

Al pulsar *Buscar*, la lista de pedidos filtrada:



☐	Número	Fecha del pedido	Entrega prometida	Cliente	Total	Estado	Estado de la factura	Actividades	
☐	S00024	30 oct, 12:39		Gemini Furniture	\$ 1.990,00	Pedido de venta	Nada que facturar		
☐	S00023	30 oct, 12:39		Gemini Furniture	\$ 4.350,00	Pedido de venta	Nada que facturar		
☐	S00022	30 oct, 12:39		Gemini Furniture	\$ 1.475,00	Pedido de venta	Nada que facturar		
☐	S00021	30 oct, 12:39		Gemini Furniture	\$ 2.175,00	Pedido de venta	Nada que facturar		
☐	S00016	30 oct, 12:39		Gemini Furniture	\$ 1.186,50	Pedido de venta	Nada que facturar	Upsell	
☐	S00014	30 oct, 12:39		Gemini Furniture	\$ 1.582,00	Pedido de venta	Nada que facturar		
☐	S00013	30 oct, 12:39		Gemini Furniture	\$ 415,50	Pedido de venta	Nada que facturar		
☐	S00012	30 oct, 12:39		Gemini Furniture	\$ 556,00	Pedido de venta	Nada que facturar	Confirm order via email	
☐	S00010	30 oct, 12:39		Gemini Furniture	\$ 751,00	Pedido de venta	Nada que facturar	Confirm Delivery	
☐	S00008	30 oct, 12:39		Gemini Furniture	\$ 462,00	Pedido de venta	Nada que facturar		
☐	S00007	30 oct, 12:39		Gemini Furniture	\$ 1.706,00	Pedido de venta	A facturar	Check delivery requirements	
☐	S00006	30 oct, 12:39		Lumber Inc	\$ 862,50	Pedido de venta	Nada que facturar		
☐	S00004	30 oct, 12:39		Gemini Furniture	\$ 2.240,00	Pedido de venta	Nada que facturar	Upsell	
☐	S00003	30 oct, 12:39		Ready Mat	\$ 434,13	Presupuesto	Nada que facturar	Answer questions	
☐	S00009	23 oct, 13:39		Gemini Furniture	\$ 654,00	Pedido de venta	Nada que facturar		

Reto

Añade un campo `state` (selección: todos / presupuestos / pedidos) que filtre por `state` del pedido.

Capítulo 15 · Ajustes propios en Configuración

Añadiremos una sección a **Ajustes** con tres opciones que cambian el comportamiento de ventas e inventario: proponer fecha de validez en presupuestos, cuántos días, y bloquear la validación de albaranes. Une los ejercicios originales `28_new_config_parameter` y `29_sale_stock_extra_configuration`.

Paso 1 · Dónde se guardan los ajustes

Los ajustes globales viven en `ir.config_parameter`, una tabla clave → valor (texto). Puedes verla en Ajustes → Técnico → Parámetros del sistema.

En Odoo 20 se leen con métodos **tipados**, que devuelven el tipo correcto y un valor por defecto:

```
ICP = self.env['ir.config_parameter'].sudo()
ICP.get_bool('ej15.propose_validity')      # True / False
ICP.get_int('ej15.validity_days', 30)      # entero, 30 si no existe
ICP.get_str('ej15.picking_lock', 'no')    # texto
ICP.set_int('ej15.validity_days', 15)
```

El original usaba `get_param` y comparaba con `'True'` como texto, una fuente de errores clásica. Además usaba claves sin prefijo (`days_ahead`) que pueden chocar con otros módulos: usa siempre `nombre_modulo.clave`.

Paso 2 · Los campos de ajustes

`models/res_config_settings.py`:

```
from odoo import fields, models

class ResConfigSettings(models.TransientModel):
    _inherit = 'res.config.settings'

    ej15_propose_validity = fields.Boolean(
        string='Proponer fecha de validez',
        config_parameter='ej15.propose_validity')
    ej15_validity_days = fields.Integer(
        string='Días de validez', default=30,
        config_parameter='ej15.validity_days')
    ej15_picking_lock = fields.Boolean(
        string='Bloquear validación de albaranes',
        config_parameter='ej15.picking_lock')
```

El atributo `config_parameter` hace todo el trabajo: lee y guarda el parámetro automáticamente. **Ya no hace falta** escribir `get_values` ni `set_values` como en el original.

Paso 3 · La vista de Ajustes (sintaxis desde Odoo 17)

Las antiguas `div` con `o_setting_box` se sustituyen por las etiquetas `<app>`, `<block>` y `<setting>`:

```
<?xml version="1.0" encoding="UTF-8"?>
<odoo>
  <record id="res_config_settings_view_form_ej15" model="ir.ui.view">
    <field name="name">res.config.settings.view.form.ej15</field>
    <field name="model">res.config.settings</field>
    <field name="inherit_id" ref="base.res_config_settings_view_form"/>
    <field name="arch" type="xml">
      <xpath expr="//form" position="inside">
        <app data-string="Extras del curso" string="Extras del curso">
```

```

name="ej15_sale_stock_config">
    <block title="Ventas" name="ej15_sale_block">
        <setting id="ej15_propose_validity"
            help="Rellena la fecha de validez de los presupuestos nuevos">
            <field name="ej15_propose_validity"/>
            <div class="mt-2" invisible="not ej15_propose_validity">
                <field name="ej15_validity_days" class="o_light_label"/> días
            </div>
        </setting>
    </block>
    <block title="Inventario" name="ej15_stock_block">
        <setting id="ej15_picking_lock"
            help="Nadie podrá validar albaranes mientras esté activo">
            <field name="ej15_picking_lock"/>
        </setting>
    </block>
</app>
</xpath>
</field>
</record>
</odoo>

```

- `<app name="...">` crea una entrada en el menú lateral de Ajustes. El `name` debe ser el nombre técnico del módulo.
- `<setting>` dibuja la casilla, la etiqueta y la ayuda; ya no hay que maquetar a mano.
- `invisible="not ej15_propose_validity"` oculta los días si la opción está apagada.

Paso 4 · Usar los ajustes en ventas

`models/sale_order.py`:

```

from datetime import timedelta

from odoo import api, fields, models

class SaleOrder(models.Model):
    _inherit = 'sale.order'

    @api.model
    def default_get(self, fields_list):
        res = super().default_get(fields_list)
        ICP = self.env['ir.config_parameter'].sudo()
        if 'validity_date' in fields_list and ICP.get_bool('ej15.propose_validity'):
            days = ICP.get_int('ej15.validity_days', 30)
            res['validity_date'] = fields.Date.context_today(self) + timedelta(days=days)
        return res

```

`default_get` se ejecuta al abrir un formulario nuevo y devuelve los valores iniciales. Comprobamos que `validity_date` se ha pedido para no añadir campos que la vista no usa.

Paso 5 · Usar los ajustes en inventario

`models/stock_picking.py`:

```

from odoo import models
from odoo.exceptions import UserError

class StockPicking(models.Model):
    _inherit = 'stock.picking'

```

```
def button_validate(self):
    if self.env['ir.config_parameter'].sudo().get_bool('ej15.picking_lock'):
        raise UserError(self.env._('La validación de albaranes está bloqueada en Ajustes.'))
    return super().button_validate()
```

Paso 6 · Manifiesto

```
{
    'name': 'Ej15 Ajustes de venta e inventario',
    'version': '20.0.1.0.0',
    'license': 'LGPL-3',
    'depends': ['sale_stock'],
    'data': ['views/res_config_settings_views.xml'],
}
```

`sale_stock` ya depende de `sale` y `stock`. No hace falta `ir.access.csv`: no creamos modelos nuevos. (El original tenía uno copiado de otro ejercicio que daba error.)

Prueba

1. Ajustes — *Extras del curso*. Activa *Proponer fecha de validez* con 15 días y guarda.
2. Crea un presupuesto: la *Fecha de validez* aparece rellena a 15 días vista.
3. Activa *Bloquear validación de albaranes*, confirma el pedido e intenta validar la entrega: aparece el aviso.

Resultado en Odoo 20

La sección *Extras del curso* en Ajustes:

Un presupuesto nuevo con el *Vencimiento* ya relleno a 15 días:

Ventas Pedidos Productos Informes Configuración
6 6 My Company (San Francisco)

Nuevo
Presupuestos
Nuevo

Enviar Descargar Confirmar Vista previa
Presupuesto Presupuesto enviado Pedido de venta

Nuevo

Ciente

Escriba para encontrar un cliente...

Vencimiento ?

14 nov

Condiciones de pago

Inmediato

Fecha de entrega ?

Líneas del pedido

Otra información

Impuestos no incluidos

Impuestos incluidos

Descripción	Cantidad	Precio unita...	Impuestos	Importe	☰
<div> Añadir línea Añadir sección Añadir nota Catálogo </div>					

Términos y condiciones...

Importe base:

\$ 0,00

Total:

\$ 0,00

Enviar mensaje
Registrar una nota
Actividad

0

Reto

Crea un asistente que lea un ajuste de tipo selección (`config_parameter` sobre un `fields.Selection`) para decidir si crea un producto o un contacto, como hacía el ejercicio 28.

Capítulo 16 · Exportar productos a Excel

Un asistente generará un `.xlsx` con código, nombre, precio y coste de los productos, y lo descargará. Ejercicio original: `16_export_product_excel`.

Qué cambia en Odoo 20

Original (Odoo 16)	Odoo 20
<code>xlwt</code> (formato <code>.xls</code> antiguo)	XlsxWriter (<code>.xlsx</code>), incluido en los requisitos de Odoo
Tipo de producto <code>product</code> (almacenable)	Tipos <code>consu</code> (bienes), <code>service</code> , <code>combo</code> ; el stock se activa con la casilla <code>is_storable</code>
Adjunto con <code>datas</code> en base64	Adjunto con <code>raw</code> en bytes

Paso 1 · El asistente

`wizard/product_export_excel.py`:

```
from io import BytesIO

import xlsxwriter

from odoo import fields, models

class ProductExportExcel(models.TransientModel):
    _name = 'product.export.excel'
    _description = 'Exportar productos a Excel'

    product_type = fields.Selection([
        ('all', 'Todos'),
        ('storable', 'Con control de stock'),
        ('consu', 'Bienes'),
        ('service', 'Servicios'),
    ], string='Tipo', default='all', required=True)

    def _get_domain(self):
        if self.product_type == 'storable':
            return [('is_storable', '=', True)]
        if self.product_type in ('consu', 'service'):
            return [('type', '=', self.product_type)]
        return []

    def action_export(self):
        self.ensure_one()
        products = self.env['product.product'].search(self._get_domain())

        buffer = BytesIO()
        workbook = xlsxwriter.Workbook(buffer, {'in_memory': True})
        sheet = workbook.add_worksheet('Productos')
        header = workbook.add_format({'bold': True, 'border': 1, 'bg_color': '#DDDDDD'})
        normal = workbook.add_format({'bottom': 1})
        warning = workbook.add_format({'bottom': 1, 'font_color': 'red', 'bold': True})
        money = workbook.add_format({'bottom': 1, 'num_format': '#,##0.00 €'})

        titles = ['Código', 'Producto', 'Precio', 'Coste']
        for col, title in enumerate(titles):
            sheet.write(0, col, title, header)
        sheet.set_column(0, 0, 14)
        sheet.set_column(1, 1, 40)
        sheet.set_column(2, 3, 12)

        for row, product in enumerate(products, start=1):
            style = normal if product.default_code else warning
            sheet.write(row, 0, product.default_code or 'SIN CÓDIGO', style)
```

```

        sheet.write(row, 1, product.display_name, style)
        sheet.write_number(row, 2, product.lst_price, money)
        sheet.write_number(row, 3, product.standard_price, money)

    workbook.close()

    attachment = self.env['ir.attachment'].create({
        'name': 'productos.xlsx',
        'raw': buffer.getvalue(),
        'mimetype': 'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet',
    })
    return {
        'type': 'ir.actions.act_url',
        'url': f'/web/content/{attachment.id}?download=true',
        'target': 'self',
    }

```

Cómo funciona:

1. `BytesIO` es un fichero en memoria: no hace falta escribir en disco.
2. `XlsxWriter` usa *formatos* para negrita, bordes, colores y números.
3. Los productos sin código salen en rojo, como en el original.
4. Se guarda como adjunto con `raw` (bytes) y se devuelve una acción `act_url` que lo descarga.

Paso 2 · Vista, acción y menú

```

<odoo>
    <record id="product_export_excel_form" model="ir.ui.view">
        <field name="name">product.export.excel.form</field>
        <field name="model">product.export.excel</field>
        <field name="arch" type="xml">
            <form>
                <group>
                    <field name="product_type" widget="radio"/>
                </group>
                <footer>
                    <button string="Exportar" name="action_export" type="object" class="btn-primary"/>
                    <button string="Cancelar" special="cancel" class="btn-secondary"/>
                </footer>
            </form>
        </field>
    </record>

    <record id="action_product_export_excel" model="ir.actions.act_window">
        <field name="name">Exportar productos a Excel</field>
        <field name="res_model">product.export.excel</field>
        <field name="view_mode">form</field>
        <field name="target">new</field>
    </record>

    <menuitem id="menu_product_export_excel" name="Exportar a Excel"
        parent="stock.menu_stock_warehouse_mgmt"
        action="action_product_export_excel" sequence="90"/>
</odoo>

```

El menú `stock.menu_stock_inventory_control` del original ya no existe; lo colgamos de Inventario — Operaciones.

Paso 3 · Permiso y manifiesto

```
id,name,model_id/group_id/id,operation,domain
```

```
access_product_export_excel,product.export.excel,product.export.excel,stock.group_stock_user,crud,
```

```
'depends': ['stock'],  
'data': ['security/ir.access.csv', 'wizard/product_export_excel_views.xml'],  
'external_dependencies': {'python': ['xlsxwriter']},
```

`external_dependencies` hace que Odoo avise si falta la librería en vez de fallar con un error confuso.

Alternativa sin código

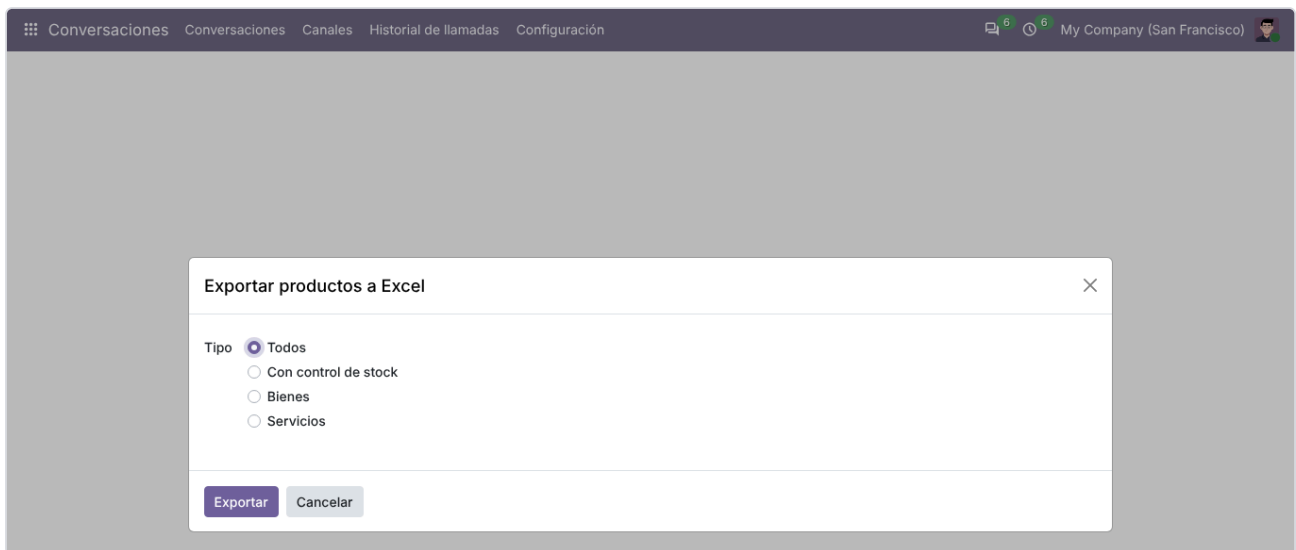
Para exportaciones puntuales, Odoo ya trae una: selecciona registros en cualquier lista → Acción → **Exportar**. Programa tu propio Excel solo cuando necesitas un formato fijo, cálculos o estilos.

Prueba

Inventario → Operaciones → *Exportar a Excel* → *Todos* → Exportar. Se descarga `productos.xlsx`; ábrelo con Excel o LibreOffice.

Resultado en Odoo 20

El asistente de exportación:



El fichero `productos.xlsx` generado, con cabecera en negrita, importes en euros y los productos sin código en rojo:

productos.xlsx — hoja «Productos» (primeras filas del fichero generado)				
	A	B	C	D
1	Código	Producto	Precio	Coste
2	DESK0005	[DESK0005] Escritorio personalizable (Blanco, Personalizado)	750,00 €	0,00 €
3	DESK0006	[DESK0006] Escritorio personalizable (Negro, Personalizado)	750,00 €	0,00 €
4	Delivery_010	[Delivery_010] Entrega local	10,00 €	0,00 €
5	E-COM06	[E-COM06] Escritorio en L con asiento a la derecha	147,00 €	600,00 €
6	E-COM07	[E-COM07] Gabinete grande	320,00 €	800,00 €
7	E-COM08	[E-COM08] Caja de almacenamiento	15,80 €	14,00 €
8	E-COM10	[E-COM10] Cubo de basura con pedal	47,00 €	10,00 €
9	E-COM11	[E-COM11] Armario con puertas	140,00 €	120,50 €
10	E-COM12	[E-COM12] Silla de conferencias (Acero)	33,00 €	0,00 €
11	E-COM13	[E-COM13] Silla de conferencias (Aluminio)	39,40 €	0,00 €
12	FURN_0001	[FURN_0001] Organizador de escritorio	5,10 €	0,00 €
13	FURN_0002	[FURN_0002] Alfombrilla de escritorio	1,98 €	0,00 €
14	FURN_0006	[FURN_0006] Soporte de monitor	3,19 €	0,00 €

Reto

Añade una columna *Stock disponible* con `product.qty_available` y una fila final con la suma del coste usando `sheet.write_formula`.

Capítulo 17 · Importar productos desde CSV y Excel

Un modelo guardará cada importación: subes el fichero, pulsas *Importar* y se crean los productos. Ejercicio original: `17_import_product`.

El fichero de ejemplo (`product.csv`) tiene esta forma:

```
Product,Code,Price,Cost
PAN,P001,10,3
AGUA,P002,15,7
```

Qué cambia en Odoo 20

- Un campo `Binary` ya no contiene base64: devuelve un objeto con `.content` (los bytes). No hace falta `base64.b64decode` ni escribir ficheros temporales.
- `xlrd` 2.0 solo lee `.xls` antiguos. Para `.xlsx` se usa **openpyxl**, también incluido en los requisitos de Odoo.
- El tipo `product` ya no existe: un producto con stock es `type='consu' + is_storable=True`.

Paso 1 · El modelo

```
import csv
import io
import os

import openpyxl

from odoo import fields, models
from odoo.exceptions import UserError

class ProductImport(models.Model):
    _name = 'product.import'
    _description = 'Importación de productos'
    _order = 'id desc'

    name = fields.Char(string='Descripción', required=True)
    file = fields.Binary(string='Fichero', required=True)
    file_name = fields.Char(string='Nombre del fichero')
    state = fields.Selection(
        [('draft', 'Borrador'), ('done', 'Importado')], default='draft', readonly=True)
    product_count = fields.Integer(string='Productos creados', readonly=True)

    # ----- lectura de ficheros -----
    def _read_csv(self):
        text = self.file.content.decode('utf-8-sig') # -sig quita el BOM de Excel
        return list(csv.DictReader(io.StringIO(text)))

    def _read_xlsx(self):
        workbook = openpyxl.load_workbook(io.BytesIO(self.file.content), read_only=True, data_only=True)
        rows = workbook.active.iter_rows(values_only=True)
        headers = [str(h).strip() for h in next(rows)]
        return [dict(zip(headers, row)) for row in rows if any(row)]

    # ----- acción -----
    def action_import(self):
        self.ensure_one()
        extension = os.path.splitext(self.file_name or '')[1].lower()
        if extension == '.csv':
            rows = self._read_csv()
        elif extension == '.xlsx':
```

```

        rows = self._read_xlsx()
    else:
        raise UserError(self.env._('Solo se admiten ficheros .csv o .xlsx.'))

    vals_list = []
    for line_number, row in enumerate(rows, start=2):
        try:
            vals_list.append({
                'name': str(row['Product']).strip(),
                'default_code': str(row['Code']).strip(),
                'list_price': float(row.get('Price') or 0),
                'standard_price': float(row.get('Cost') or 0),
                'type': 'consu',
                'is_storable': True,
            })
        except (KeyError, ValueError, TypeError) as error:
            raise UserError(self.env._(
                'Error en la línea %(line)s: %(error)s', line=line_number, error=error))

    self.env['product.template'].create(vals_list)
    self.write({'state': 'done', 'product_count': len(vals_list)})

```

Mejoras sobre el original:

1. Un único `create(vals_list)` crea todos los productos de golpe (mucho más rápido).
2. El `except:` vacío del original ocultaba el error real y además el mensaje estaba mal formateado. Aquí se captura cada tipo de error y se indica la línea exacta.
3. Si falla una línea, Odoo deshace toda la importación: no quedan productos a medias.
4. El tipo de fichero se deduce de la extensión; ya no hace falta elegirlo.

Paso 2 · Vistas

```

<odoo>
<record id="product_import_list" model="ir.ui.view">
    <field name="name">product.import.list</field>
    <field name="model">product.import</field>
    <field name="arch" type="xml">
        <list>
            <field name="name"/>
            <field name="file_name"/>
            <field name="product_count"/>
            <field name="state" widget="badge" decoration-success="state == 'done'"/>
        </list>
    </field>
</record>

<record id="product_import_form" model="ir.ui.view">
    <field name="name">product.import.form</field>
    <field name="model">product.import</field>
    <field name="arch" type="xml">
        <form>
            <header>
                <button name="action_import" type="object" string="Importar"
                    class="btn-primary" invisible="state == 'done'"/>
                <field name="state" widget="statusbar"/>
            </header>
            <sheet>
                <group>
                    <group>
                        <field name="name" readonly="state == 'done'"/>
                        <field name="product_count" invisible="state == 'draft'"/>
                    </group>
                </group>
            </sheet>
        </form>
    </field>
</record>

```

```

        <field name="file" filename="file_name" readonly="state == 'done'"/>
        <field name="file_name" invisible="1"/>
      </group>
    </group>
  </sheet>
</form>
</field>
</record>

<record id="action_product_import" model="ir.actions.act_window">
  <field name="name">Importar productos</field>
  <field name="res_model">product.import</field>
  <field name="view_mode">list,form</field>
</record>

<menuitem id="menu_product_import" name="Importar productos"
  parent="stock.menu_stock_warehouse_mgmt"
  action="action_product_import" sequence="91"/>
</odoo>

```

Fíjate en el paso de `attrs` a expresiones directas: `invisible="state == 'done'"`, `readonly="state == 'done'"`. El original además usaba `<openerp>` como raíz, que hace años que no se admite: siempre `<odoo>`.

Paso 3 · Permiso y manifiesto

```

id,name,model_id,group_id/id,operation,domain
access_product_import,product.import,product.import,stock.group_stock_manager,crud,

```

```

'depends': ['stock'],
'data': ['security/ir.access.csv', 'views/product_import_views.xml'],
'external_dependencies': {'python': ['openpyxl']},

```

La alternativa estándar

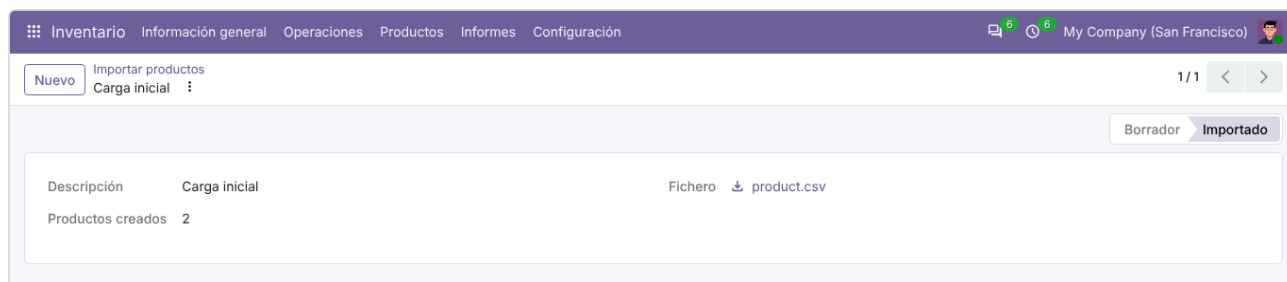
Odoo trae un importador genérico: en cualquier lista, Favoritos — **Importar registros**. Admite CSV y Excel y empareja columnas. Programa tu propio importador solo si hay reglas de negocio (validaciones, cálculos, varias tablas a la vez).

Prueba

1. Inventario — Operaciones — *Importar productos* — Nuevo.
2. Sube `product.csv` del repositorio y pulsa *Importar*.
3. Comprueba los productos PAN y AGUA en Inventario — Productos.
4. Repite con `product_excel.xlsx` (guárdalo antes como `.xlsx` si fuera `.xls`).

Resultado en Odoo 20

La importación de `product.csv` en estado *Importado*, con 2 productos creados:



Los productos PAN y AGUA ya en Inventario:

Inventario Información general Operaciones Productos Informes Configuración					
Nuevo Productos					
Bienes x Producto P00 x Buscar...					
1-2 / 2					
☐	Nombre del producto	Referencia interna	Precio de venta	Coste	Stock real
☐	☆ AGUA	P002	\$ 15,00	\$ 7,00	0,00
☐	☆ PAN	P001	\$ 10,00	\$ 3,00	0,00
					0,00

Reto

Si ya existe un producto con el mismo `default_code`, actualiza su precio en lugar de crear uno nuevo (usa `search` antes de `create`).

Parte V · Contabilidad

Capítulo 18 · Crear una factura desde código

Un asistente pedirá cliente, producto, cantidad y precio, creará la factura en borrador y la abrirá. Es la base de cualquier integración que facture automáticamente. Ejercicio original: `19_create_invoice`.

Cómo es una factura en Odoo

Todo lo contable es un **asiento** (`account.move`). Lo que cambia es `move_type`:

move_type	Documento
out_invoice	Factura de cliente
out_refund	Factura rectificativa de cliente
in_invoice	Factura de proveedor
in_refund	Rectificativa de proveedor
entry	Asiento manual (capítulo 19)

En una factura solo das las **líneas de producto** (`invoice_line_ids`). Odoo calcula solo las líneas de impuestos y la del cliente por cobrar.

Paso 1 · El asistente

```
from odoo import fields, models

class CreateInvoiceWizard(models.TransientModel):
    _name = 'create.invoice.wzd'
    _description = 'Crear factura de cliente'

    company_id = fields.Many2one(
        'res.company', string='Empresa', default=lambda self: self.env.company, required=True)
    journal_id = fields.Many2one(
        'account.journal', string='Diario',
        domain="[('type', '=', 'sale'), ('company_id', '=', company_id)]")
    partner_id = fields.Many2one('res.partner', string='Cliente', required=True)
    invoice_date = fields.Date(string='Fecha', required=True, default=fields.Date.context_today)
    payment_term_id = fields.Many2one('account.payment.term', string='Plazo de pago')
    product_id = fields.Many2one('product.product', string='Producto', required=True)
    quantity = fields.Float(string='Cantidad', default=1.0)
    price_unit = fields.Float(string='Precio unitario')
    analytic_account_id = fields.Many2one('account.analytic.account', string='Cuenta analítica')
    notes = fields.Text(string='Notas')

    def _prepare_invoice_line_vals(self):
        vals = {
            'product_id': self.product_id.id,
            'quantity': self.quantity,
        }
        if self.price_unit:
            vals['price_unit'] = self.price_unit # si no, Odoo usa el precio del producto
        if self.analytic_account_id:
            vals['analytic_distribution'] = {str(self.analytic_account_id.id): 100}
        return vals

    def _prepare_invoice_vals(self):
        vals = {
```

```

        'move_type': 'out_invoice',
        'partner_id': self.partner_id.id,
        'company_id': self.company_id.id,
        'invoice_date': self.invoice_date,
        'narration': self.notes,
        'invoice_line_ids': [fields.Command.create(self._prepare_invoice_line_vals())],
    }
    if self.journal_id:
        vals['journal_id'] = self.journal_id.id
    if self.payment_term_id:
        vals['invoice_payment_term_id'] = self.payment_term_id.id
    return vals

def action_create_invoice(self):
    self.ensure_one()
    invoice = self.env['account.move'].create(self._prepare_invoice_vals())
    return {
        'type': 'ir.actions.act_window',
        'res_model': 'account.move',
        'res_id': invoice.id,
        'view_mode': 'form',
        'target': 'current',
    }

```

Qué ha cambiado respecto al original y por qué:

- **Menos campos que rellenar a mano.** Odoo deduce la moneda, la posición fiscal, la cuenta de ingresos, los impuestos y la descripción a partir del cliente y el producto. Pasarlos todos, como hacía el original, puede pisar esa lógica.
- `fields.Command.create(vals)` sustituye a la tupla mágica `(0, 0, vals)`. Ambas funcionan, pero la primera se lee mejor.
- La acción de retorno se construye a mano: es más corta y no depende de ids externos.
- `analytic_distribution` es un diccionario `{id_cuenta: porcentaje}`: puedes repartir una línea entre varias cuentas analíticas.

Paso 2 · Vista y acción

```

<odoo>
<record id="create_invoice_wzd_form" model="ir.ui.view">
    <field name="name">create.invoice.wzd.form</field>
    <field name="model">create.invoice.wzd</field>
    <field name="arch" type="xml">
        <form>
            <group>
                <group string="Factura">
                    <field name="partner_id"/>
                    <field name="invoice_date"/>
                    <field name="journal_id" options="{ 'no_create': True }"/>
                    <field name="payment_term_id"/>
                    <field name="company_id" invisible="1"/>
                </group>
                <group string="Línea">
                    <field name="product_id"/>
                    <field name="quantity"/>
                    <field name="price_unit"/>
                    <field name="analytic_account_id" groups="analytic.group_analytic_accounting"/>
                </group>
            </group>
            <field name="notes" placeholder="Notas para el cliente..."/>
        </form>
        <button string="Crear factura" name="action_create_invoice" type="object" class="btn-
primary"/>
    </field>
</record>

```

```

        <button string="Cancelar" special="cancel" class="btn-secondary"/>
      </footer>
    </form>
  </field>
</record>

<record id="action_create_invoice_wzd" model="ir.actions.act_window">
  <field name="name">Factura rápida</field>
  <field name="res_model">create.invoice.wzd</field>
  <field name="view_mode">form</field>
  <field name="target">new</field>
</record>

<menuitem id="menu_create_invoice_wzd" name="Factura rápida"
  parent="account.menu_finance_receivables"
  action="action_create_invoice_wzd" sequence="99"/>
</odoo>

```

Permiso:

```

id,name,model_id,group_id/id,operation,domain
access_create_invoice_wzd,create.invoice.wzd,create.invoice.wzd,account.group_account_invoice,crud,

```

Manifiesto: `'depends': ['account']`.

Paso 3 · Confirmar la factura

La factura se crea en **borrador**. Para confirmarla desde código:

```
invoice.action_post()
```

El original del capítulo 19 usaba `move.post()`, que dejó de existir hace varias versiones.

Prueba

Facturación — Clientes — *Factura rápida*. Elige un cliente, un producto y 3 unidades. Se abre la factura en borrador con impuestos calculados. Pulsa *Confirmar* para darle número.

Resultado en Odoo 20

El asistente *Factura rápida*:

La factura en borrador que crea, con impuestos y total calculados por Odoo:

Reto

Permite varias líneas: cambia los campos de producto por un `One2many` a un modelo transitorio `create.invoice.wzd.line` y crea una línea de factura por cada una.

Capítulo 19 · Crear un asiento contable

Desde la ficha de un producto, dos botones crearán un asiento de ingreso o de gasto por su importe, y un botón inteligente mostrará los asientos generados. Ejercicio original: `20_create_move`.

La regla de oro

Un asiento tiene al menos dos líneas y **la suma del debe tiene que ser igual a la del haber**. Si no cuadra, Odoo no lo guarda.

Línea	Cuenta	Debe	Haber
1	705 Prestaciones de servicios (ingreso)	0	100
2	430 Clientes	100	0

Paso 1 · El código

```
from odoo import fields, models
from odoo.exceptions import UserError

class ProductTemplate(models.Model):
    _inherit = 'product.template'

    account_entry_ids = fields.Many2many(
        'account.move', 'product_template_account_move_rel', string='Asientos', copy=False)
    account_entry_count = fields.Integer(compute='_compute_account_entry_count')

    def _compute_account_entry_count(self):
        for product in self:
            product.account_entry_count = len(product.account_entry_ids)

    # ----- utilidades -----
    def _get_general_journal(self):
        journal = self.env['account.journal'].search([
            ('type', '=', 'general'),
            ('company_id', '=', self.env.company.id),
        ], limit=1)
        if not journal:
            raise UserError(self.env._('Crea un diario de tipo Varios para esta empresa.'))
        return journal

    def _get_accounts(self):
        product = self.with_company(self.env.company)
        income = product.property_account_income_id or product.categ_id.property_account_income_categ_id
        expense = product.property_account_expense_id or
        product.categ_id.property_account_expense_categ_id
        if not income or not expense:
            raise UserError(self.env._('Define las cuentas de ingresos y gastos del producto o de su categoría.'))
        return income, expense

    def _create_move(self, amount, debit_account, credit_account):
        self.ensure_one()
        if amount <= 0:
            raise UserError(self.env._('El importe debe ser mayor que cero.'))
        move = self.env['account.move'].create({
            'move_type': 'entry',
            'journal_id': self._get_general_journal().id,
            'date': fields.Date.context_today(self),
            'ref': self.name,
            'line_ids': [
                fields.Command.create({'name': self.name, 'account_id': debit_account.id, 'debit':
amount}),
```

```

        fields.Command.create({'name': self.name, 'account_id': credit_account.id, 'credit':
amount})),
    ],
    })
    move.action_post()
    self.account_entry_ids = [fields.Command.link(move.id)]
    return move

# ----- botones -----
def action_make_move_income(self):
    income, expense = self._get_accounts()
    self._create_move(self.list_price, debit_account=expense, credit_account=income)

def action_make_move_expense(self):
    income, expense = self._get_accounts()
    cost = self.with_company(self.env.company).standard_price
    self._create_move(cost, debit_account=expense, credit_account=income)

def action_view_move_entries(self):
    self.ensure_one()
    return {
        'type': 'ir.actions.act_window',
        'name': self.env._('Asientos'),
        'res_model': 'account.move',
        'view_mode': 'list,form',
        'domain': [('id', 'in', self.account_entry_ids.ids)],
        'context': {'create': False},
    }

```

Correcciones respecto al original:

1. `move.post()` ya no existe: es `action_post()`.
2. Las líneas de un asiento manual **no** llevan `display_type='product'` (eso es para facturas).
3. Si el producto no tiene cuentas propias se usan las de su categoría, que es lo habitual.
4. Se valida que el importe sea positivo: un asiento a cero no tiene sentido.
5. El original duplicaba los métodos en `product.product` solo para lanzar un error; no hace falta.

En un caso real usarías un **banco** o **clientes** como contrapartida, no ingreso contra gasto. Aquí se mantiene la lógica del ejercicio original para centrarnos en la técnica.

Paso 2 · Botones en la ficha del producto

```

<odoo>
<record id="product_template_form_moves" model="ir.ui.view">
  <field name="name">product.template.form.moves</field>
  <field name="model">product.template</field>
  <field name="inherit_id" ref="product.product_template_form_view"/>
  <field name="arch" type="xml">
    <xpath expr="//sheet" position="before">
      <header>
        <button name="action_make_move_income" type="object"
          string="Asiento de ingreso" class="btn-primary"
          groups="account.group_account_manager"/>
        <button name="action_make_move_expense" type="object"
          string="Asiento de gasto"
          groups="account.group_account_manager"/>
      </header>
    </xpath>
    <div name="button_box" position="inside">
      <button name="action_view_move_entries" type="object"
        class="oe_stat_button" icon="receipt_long"
        invisible="not account_entry_count">

```

```

        <field name="account_entry_count" string="Asientos" widget="statinfo"/>
    </button>
</div>
</field>
</record>
</odoo>

```

Si el formulario de producto ya tiene un `<header>` en tu instalación, cambia la primera `xpath` por `//header` con `position="inside"`.

Manifiesto: `'depends': ['account']`.

Prueba

1. Facturación — Configuración — Diarios: comprueba que existe un diario *Operaciones varias*.
2. Abre un producto con precio de venta 100 € y pulsa *Asiento de ingreso*.
3. Aparece el botón **Asientos 1**. Ábrelo: el asiento está publicado y cuadrado.

Resultado en Odoo 20

Los botones *Asiento de ingreso* y *Asiento de gasto* y el botón inteligente **Asientos**:

El asiento publicado en el diario de operaciones varias:

Facturación

Cientes

Proveedores

Informes

Configuración

6

6

My Company (San Francisco)

6

Nuevo

Asientos contables

Varios/2026/10/0003 (Escritorio pe...

1/1

<

>

Revertir asiento

Restablecer a borrador

Borrador

Registrado

Varios/2026/10/0003

Referencia

Escritorio personalizable

Fecha contable

30 oct

Diario

Operaciones varias

Otra información

Contabilizar automáticamente

No

Posición fiscal

Compañía

My Company (San Francisco)

Añada una nota interna...

Enviar mensaje

Registrar una nota

Actividad

Hoy

OdooBot

12:44

Asiento contable creado

Reto

Añade un asistente que pida el importe y la cuenta de contrapartida (por ejemplo, el banco) en vez de usar siempre ingreso contra gasto.

Capítulo 20 · Factura, pago y conciliación

Un documento propio recorrerá tres estados con tres botones: *Crear factura* → *Registrar pago* → factura **Pagada**. Ejercicio original: `21_create_invoice_and_payment`.

El flujo

1. **Factura publicada:** el cliente debe 121 € (cuenta de clientes al debe).
2. **Pago publicado:** entra dinero en el banco; el cliente queda con 121 € a favor.
3. **Conciliación:** Odoo enlaza la deuda de la factura con el pago y la factura pasa a *Pagada* (o *En proceso de pago* si el banco aún no lo ha confirmado).

El original creaba el pago a mano y después conciliaba con `js_assign_outstanding_line` dentro de un `try/except: pass` que escondía cualquier error. En Odoo 20 lo correcto es usar el **asistente de registro de pagos** (`account.payment.register`), el mismo que usa el botón *Registrar pago*: crea el pago y **concilia** en un solo paso.

Paso 1 · El modelo

```
from odoo import api, fields, models
from odoo.exceptions import UserError

class CreateInvoiceModel(models.Model):
    _name = 'create.invoice.model'
    _description = 'Factura y cobro'
    _order = 'id desc'

    name = fields.Char(string='Número', default='New', readonly=True, copy=False)
    company_id = fields.Many2one(
        'res.company', string='Empresa', default=lambda self: self.env.company, required=True)
    partner_id = fields.Many2one('res.partner', string='Cliente', required=True)
    product_id = fields.Many2one('product.product', string='Producto', required=True)
    quantity = fields.Float(string='Cantidad', default=1.0)
    price_unit = fields.Float(string='Precio unitario')
    date_invoice = fields.Date(string='Fecha', required=True, default=fields.Date.context_today)
    payment_journal_id = fields.Many2one(
        'account.journal', string='Diario de cobro',
        domain="[('type', 'in', ('bank', 'cash')), ('company_id', '=', company_id)]")
    state = fields.Selection([
        ('draft', 'Borrador'),
        ('invoiced', 'Facturado'),
        ('paid', 'Cobrado'),
    ], default='draft', readonly=True)
    invoice_id = fields.Many2one('account.move', string='Factura', readonly=True, copy=False)
    payment_state = fields.Selection(related='invoice_id.payment_state', string='Estado de pago')

    @api.model_create_multi
    def create(self, vals_list):
        for vals in vals_list:
            if vals.get('name', 'New') == 'New':
                vals['name'] = self.env['ir.sequence'].next_by_code('create.invoice.model') or 'New'
        return super().create(vals_list)

    def action_create_invoice(self):
        self.ensure_one()
        line_vals = {'product_id': self.product_id.id, 'quantity': self.quantity}
        if self.price_unit:
            line_vals['price_unit'] = self.price_unit
        invoice = self.env['account.move'].create({
            'move_type': 'out_invoice',
            'partner_id': self.partner_id.id,
```

```

        'company_id': self.company_id.id,
        'invoice_date': self.date_invoice,
        'invoice_line_ids': [fields.Command.create(line_vals)],
    })
    invoice.action_post()
    self.write({'invoice_id': invoice.id, 'state': 'invoiced'})

def action_register_payment(self):
    self.ensure_one()
    if not self.payment_journal_id:
        raise UserError(self.env._('Elige un diario de cobro.'))
    wizard = self.env['account.payment.register'].with_context(
        active_model='account.move',
        active_ids=self.invoice_id.ids,
    ).create({
        'journal_id': self.payment_journal_id.id,
        'payment_date': fields.Date.context_today(self),
    })
    wizard._create_payments()      # crea, publica y concilia
    self.state = 'paid'

def action_open_invoice(self):
    self.ensure_one()
    return {
        'type': 'ir.actions.act_window',
        'res_model': 'account.move',
        'res_id': self.invoice_id.id,
        'view_mode': 'form',
    }

```

Por qué así:

- **El asistente conoce todas las reglas:** método de pago (`payment_method_line_id`, que substituyó a `payment_method_id`), moneda, importe pendiente, pagos parciales, diferencias de cambio. Reescribir todo eso a mano es fuente de errores.
- `active_model` y `active_ids` en el contexto simulan que el usuario seleccionó la factura y pulsó *Registrar pago*.
- En Odoo 18+ el campo `ref` del pago se llama `memo` ; con el asistente no necesitas tocarlo.
- `payment_state` relacionado muestra en tu documento si la factura está pagada.

Paso 2 · Formulario con barra de estado

```

<record id="create_invoice_model_form" model="ir.ui.view">
  <field name="name">create.invoice.model.form</field>
  <field name="model">create.invoice.model</field>
  <field name="arch" type="xml">
    <form>
      <header>
        <button name="action_create_invoice" type="object" string="Crear factura"
          class="btn-primary" invisible="state != 'draft'"/>
        <button name="action_register_payment" type="object" string="Registrar pago"
          class="btn-primary" invisible="state != 'invoiced'"/>
        <field name="state" widget="statusbar"/>
      </header>
      <sheet>
        <div name="button_box" class="oe_button_box">
          <button name="action_open_invoice" type="object" class="oe_stat_button"
            icon="receipt_long" invisible="not invoice_id" string="Factura"/>
        </div>
        <div class="oe_title"><h1><field name="name"/></h1></div>
        <group>
          <group>
            <field name="partner_id" readonly="state != 'draft'"/>
            <field name="date_invoice" readonly="state != 'draft'"/>

```

```

        <field name="payment_journal_id" readonly="state == 'paid'"/>
        <field name="company_id" invisible="1"/>
    </group>
    <group>
        <field name="product_id" readonly="state != 'draft'"/>
        <field name="quantity" readonly="state != 'draft'"/>
        <field name="price_unit" readonly="state != 'draft'"/>
        <field name="invoice_id" invisible="not invoice_id"/>
        <field name="payment_state" widget="badge" invisible="not invoice_id"/>
    </group>
</group>
</sheet>
</form>
</field>
</record>

```

Completa el módulo con lo que ya sabes: vista lista, acción, menú (por ejemplo bajo `account.menu_finance_receivables`), la secuencia `create.invoice.model` (capítulo 6) y el permiso:

```

id,name,model_id,group_id/id,operation,domain
access_create_invoice_model,create.invoice.model,create.invoice.model,account.group_account_invoice,crud,

```

Manifiesto: `'depends': ['account']`.

Prueba

1. Crea un documento: cliente, producto, cantidad 1 y diario *Banco*.
2. *Crear factura*: el estado pasa a *Facturado* y aparece el botón **Factura**.
3. *Registrar pago*: el estado pasa a *Cobrado* y el estado de pago muestra *Pagado* o *En proceso de pago*.
4. Abre la factura: en la parte inferior verás el pago aplicado.

Resultado en Odoo 20

Un documento facturado y pendiente de cobro:

The screenshot displays the Odoo 20 web interface for a 'Factura y cobro' (Invoice and payment) document. The document ID is FC/0002. The top navigation bar includes 'Facturación', 'Clientes', 'Proveedores', 'Informes', and 'Configuración'. The document is currently in the 'Facturado' (Invoiced) state, with buttons for 'Borrador' (Draft), 'Facturado', and 'Cobrado' (Paid). The document details are as follows:

Factura y cobro		FC/0002	
Cliente	Gemini Furniture	Producto	[DESK0005] Escritorio personalizable (Blanco, Personalizado)
Fecha	30 oct	Cantidad	1,00
Diario de cobro	Banco	Precio unitario	0,00
		Factura	INV/2026/00011
		Estado de pago	No pagado

Tras *Registrar pago*: estado *Cobrado* y estado de pago *Pagado*:

Facturación Clientes Proveedores Informes Configuración 6 6 My Company (San Francisco)

Nuevo Factura y cobro FC/0001 Factura 1/1

Borrador Facturado **Cobrado**

FC/0001

Cliente	Acme Corporation	Producto	[DESK0005] Escritorio personalizable (Blanco, Personalizado)
Fecha	30 oct	Cantidad	2,00
Diario de cobro	Banco	Precio unitario	0,00
		Factura	INV/2026/00010
		Estado de pago	Pagado

La factura queda marcada como pagada, con el pago aplicado:

Facturación Clientes Proveedores Informes Configuración 6 6 My Company (San Francisco)

Nuevo Facturas INV/2026/00010 Pagos 1 Apuntes contables 1/1

Enviar Imprimir Vista previa Factura rectificativa Restablecer a borrador Borrador Registrado

Factura de cliente
INV/2026/00010

Cliente Acme Corporation 77 Santa Barbara Rd Pleasant Hill CA 94523 Estados Unidos US12345673	Fecha de factura 30 oct Condiciones de pago 30 días
---	--

Líneas de factura Otra Información Impuestos excl.

Producto	Cantidad	Precio	Impuestos	Importe
[DESK0005] Escritorio personalizable (Blanco, Personalizado) 160 x 80 cm, con patas grandes	2,00	750,00	15%	\$ 1.500,00

Términos y condiciones

Importe base:	\$ 1.500,00
Tax 15%:	\$ 225,00
Total:	\$ 1.725,00
Pagado el 30/10/2026	\$ 1.725,00
Importe adeudado:	\$ 0,00

Reto

Añade un campo `amount` para permitir cobros parciales: pásalo al asistente como `'amount': self.amount` y comprueba que la factura queda *Pagada parcialmente*.

Parte VI · Integraciones

Capítulo 21 · Leer y escribir en Odoo desde fuera: API JSON-2

Un script Python externo, que no forma parte de Odoo, cargará contactos desde un CSV y después los modificará. Es lo que hace una tienda web, un TPV o un programa de terceros para hablar con Odoo. Une los ejercicios originales

`10_cargar_datos_xml_rpc` y `11_modificar_datos_xml_rpc`.

XML-RPC está obsoleto

En Odoo 20 los puntos `/xmlrpc`, `/xmlrpc/2` y `/jsonrpc` siguen funcionando, pero el propio código de Odoo avisa en el log de que están **obsoletos desde Odoo 19 y se eliminarán en Odoo 22**. Los scripts del repositorio funcionan hoy, pero no escribas integraciones nuevas así.

Lo nuevo es la **API JSON-2**:

	XML-RPC (antiguo)	JSON-2 (Odoo 19+)
URL	<code>/xmlrpc/2/object</code>	<code>/json/2/<modelo>/<método></code>
Autenticación	Usuario + contraseña en cada llamada	Clave API en la cabecera <code>Authorization: bearer</code>
Formato	XML	JSON normal
Argumentos	Posicionales, en listas anidadas	Por nombre: <code>domain</code> , <code>fields</code> , <code>vals_list</code> ...
Errores	Excepciones XML-RPC poco claras	Códigos HTTP (401, 404, 422) con mensaje

Paso 1 · Crea una clave API

1. Arriba a la derecha, tu avatar → **Mi perfil** → pestaña *Seguridad de la cuenta*.
2. *Nueva clave API*, ponle nombre (`script-curso`) y duración.
3. Copia la clave: solo se muestra una vez. Trátala como una contraseña.

Paso 2 · Un cliente mínimo

Instala `requests` (`pip install requests`) y crea `odoo_client.py`:

```
import requests

ODOO_URL = 'http://localhost:8069'
DATABASE = 'curso20'
API_KEY = 'pega-aquí-tu-clave'

def call(model, method, **params):
    """Llama a un método de un modelo de Odoo con la API JSON-2."""
    response = requests.post(
        f'{ODOO_URL}/json/2/{model}/{method}',
        headers={
            'Authorization': f'bearer {API_KEY}',
            'X-Odoo-Database': DATABASE, # necesario si el servidor tiene varias bases
        },
        json=params,
        timeout=30,
    )
```

```

if not response.ok:
    raise RuntimeError(f'{response.status_code}: {response.text}')
return response.json()

```

Los parámetros se pasan **por nombre**, igual que en Python dentro de Odoo. `ids` indica sobre qué registros se ejecuta el método.

Paso 3 · Cargar contactos desde CSV

`data.csv`:

```

name,email,phone
Juan,juan@demo.cl,+3412345678
Pedro,pedro@demo.cl,+3487654321

```

`cargar_contactos.py`:

```

import csv

from odoo_client import call

with open('data.csv', encoding='utf-8') as f:
    rows = [{k: v.strip() for k, v in row.items()} for row in csv.DictReader(f)]

names = [row['name'] for row in rows]
existing = call('res.partner', 'search_read',
               domain=[('name', 'in', names)], fields=['name'])
existing_names = {p['name'] for p in existing}

new_rows = [row for row in rows if row['name'] not in existing_names]
if new_rows:
    ids = call('res.partner', 'create', vals_list=new_rows)
    print(f'Creados {len(ids)} contactos: {ids}')
print(f'Ya existían {len(existing_names)}')

```

Cambios respecto al original: una sola búsqueda para todos los nombres y un solo `create` con la lista completa, en vez de dos llamadas por fila. Con 1.000 filas pasa de 2.000 peticiones a 2.

Paso 4 · Modificar contactos

`modificar_contactos.py`:

```

import csv

from odoo_client import call

with open('data.csv', encoding='utf-8') as f:
    for row in csv.DictReader(f):
        name = row['name'].strip()
        ids = call('res.partner', 'search', domain=[('name', '=', name)])
        if not ids:
            print(f'No existe: {name}')
            continue
        call('res.partner', 'write', ids=ids,
            vals={'email': row['email'].strip(), 'phone': row['phone'].strip()})
        print(f'Actualizado: {name}')

```

Paso 5 · Probarlo con curl

```
curl -X POST http://localhost:8069/json/2/res.partner/search_read \
-H "Authorization: bearer TU_CLAVE" \
-H "X-Odoo-Database: curso20" \
-H "Content-Type: application/json" \
-d '{"domain": [["is_company", "=", true]], "fields": ["name", "email"], "limit": 5}'
```

Nota: en JSON las tuplas del dominio se escriben como listas.

Si necesitas mantener XML-RPC

Para un script antiguo que todavía no puedes migrar, sustituye la contraseña por una **clave API** (funciona igual en `login` y `execute_kw`) y cambia `execute(...)` por `execute_kw(...)` con argumentos en lista. Planifica la migración antes de Odoo 22.

Errores frecuentes

Código	Causa
401	Clave API incorrecta o caducada
404	Modelo o método inexistente, o nombre de base incorrecto
422	Argumento con nombre equivocado (por ejemplo <code>vals</code> en <code>create</code> , que espera <code>vals_list</code>)
403	El usuario de la clave no tiene permiso sobre ese modelo

Prueba

Ejecuta `python cargar_contactos.py`: Juan y Pedro aparecen en Contactos. Ejecútalo otra vez: dice que ya existían. Cambia un teléfono en el CSV y ejecuta `modificar_contactos.py`.

Resultado en Odoo 20

La salida real de los tres scripts contra Odoo 20:

```
Terminal — scripts JSON-2

$ python cargar_contactos.py
Creados 2 contactos: [66, 67]
Ya existían 0
$ python cargar_contactos.py
Ya existían 2
$ python modificar_contactos.py
Actualizado: Juan
Actualizado: Pedro
```

Juan y Pedro en Contactos, con el teléfono ya actualizado:

Contactos						Configuración		My Company (San Francisco)	
Nuevo		Contactos		Correo electrónico demo.cl		Buscar...		1-2 / 2	
☐	Nombre	Correo electrónico	Teléfono	Actividades	Pais				
☐	Juan	juan@demo.cl	+34600000001						
☐	Pedro	pedro@demo.cl	+3487654321						

Reto

Escribe `exportar_ventas.py`, que guarde en un CSV el número, cliente y total de los pedidos confirmados (`sale.order`, dominio `[('state', '=', 'sale')]`).

Capítulo 22 · Subir ficheros a un FTP con una tarea programada

Primero un asistente para subir un fichero a mano; después una **tarea programada** (cron) que cada noche genera un CSV de contactos y lo sube sola. Ejercicio original: `12_ftp`.

Paso 1 · Parámetros de conexión

`data/ftp_parameters.xml` (en `data`, con `noupdate` para no pisar los valores reales al actualizar):

```
<odoo noupdate="1">
  <record id="param_ftp_host" model="ir.config_parameter">
    <field name="key">ej22_ftp.host</field>
    <field name="value">127.0.0.1</field>
  </record>
  <record id="param_ftp_port" model="ir.config_parameter">
    <field name="key">ej22_ftp.port</field>
    <field name="value">21</field>
  </record>
  <record id="param_ftp_user" model="ir.config_parameter">
    <field name="key">ej22_ftp.user</field>
    <field name="value">curso</field>
  </record>
  <record id="param_ftp_folder" model="ir.config_parameter">
    <field name="key">ej22_ftp.folder</field>
    <field name="value">/subidas</field>
  </record>
</odoo>
```

La contraseña **no** va en un fichero del módulo (acabaría en Git). Créala a mano en Ajustes → Técnico → Parámetros del sistema con la clave `ej22_ftp.password`, o mejor aún, usa SFTP con clave pública.

Paso 2 · Un servicio reutilizable

Un `AbstractModel` no tiene tabla: sirve para agrupar lógica que usan otros modelos.

`models/ftp_service.py`:

```
import csv
import ftplib
import io

from odoo import fields, models
from odoo.exceptions import UserError

class FtpService(models.AbstractModel):
    _name = 'ej22.ftp.service'
    _description = 'Servicio FTP'

    def upload(self, filename, content: bytes):
        ICP = self.env['ir.config_parameter'].sudo()
        host = ICP.get_str('ej22_ftp.host')
        if not host:
            raise UserError(self.env._('Configura el parámetro ej22_ftp.host.'))
        folder = ICP.get_str('ej22_ftp.folder', '/')
        try:
            with ftplib.FTP() as ftp:
                ftp.connect(host, ICP.get_int('ej22_ftp.port', 21), timeout=30)
                ftp.login(ICP.get_str('ej22_ftp.user'), ICP.get_str('ej22_ftp.password'))
                ftp.cwd(folder)
```

```

        ftp.storbinary(f'STOR {filename}', io.BytesIO(content))
    except ftplib.all_errors as error:
        raise UserError(self.env._('Error FTP: %(error)s', error=error)) from error

def cron_upload_partners(self):
    """Llamado por la tarea programada."""
    buffer = io.StringIO()
    writer = csv.writer(buffer)
    writer.writerow(['id', 'nombre', 'email', 'telefono'])
    for partner in self.env['res.partner'].search([('is_company', '=', True)]):
        writer.writerow([partner.id, partner.name, partner.email or '', partner.phone or ''])
    filename = f'contactos_{fields.Date.today():%Y%m%d}.csv'
    self.upload(filename, buffer.getvalue().encode('utf-8'))

```

Correcciones respecto al original:

- El original intentaba abrir un fichero temporal que **nunca escribía**, así que siempre fallaba. Ahora el contenido va directo desde memoria con `io.BytesIO`.
- `with ftplib.FTP()` as `ftp` cierra la conexión aunque haya un error.
- `ftplib.all_errors` captura los errores de red y FTP; el resto de errores no se esconden.
- Se usan los getters tipados (`get_str`, `get_int`) de Odoo 20.

Paso 3 · El asistente manual

```

from odoo import fields, models

class FtpUploadWizard(models.TransientModel):
    _name = 'ej22.ftp.upload.wzd'
    _description = 'Subir fichero a FTP'

    file = fields.Binary(string='Fichero', required=True)
    file_name = fields.Char(string='Nombre')

    def action_upload(self):
        self.ensure_one()
        self.env['ej22.ftp.service'].upload(self.file_name, self.file.content)
        return {
            'type': 'ir.actions.client',
            'tag': 'display_notification',
            'params': {'message': self.env._('Fichero subido'), 'type': 'success', 'next': {'type':
'ir.actions.act_window_close'}}},
        }

```

`self.file.content` devuelve los bytes del fichero (Odoo 20). La acción `display_notification` muestra un aviso verde y cierra el diálogo.

Paso 4 · La tarea programada

`data/ir_cron.xml`:

```

<odoo noupdate="1">
  <record id="cron_upload_partners_ftp" model="ir.cron">
    <field name="name">Subir contactos al FTP</field>
    <field name="model_id" ref="model_ej22_ftp_service"/>
    <field name="state">code</field>
    <field name="code">model.cron_upload_partners()</field>
    <field name="interval_number">1</field>
    <field name="interval_type">days</field>
    <field name="nextcall" eval="(DateTime.now() + timedelta(days=1)).strftime('%Y-%m-%d 02:00:00')"/>
  </record>
</odoo>

```

```
<field name="active" eval="False"/>
</record>
</odoo>
```

- `code` ejecuta Python; `model` es el modelo indicado en `model_id`.
- `numbercall` y `doall`, que verás en tutoriales antiguos, ya no existen: un cron se repite hasta que lo desactivas.
- Se crea **desactivado** para que no intente conectar hasta que configures el FTP.

Paso 5 · Manifiesto y permisos

```
'depends': ['base'],
'data': [
    'security/ir.access.csv',
    'data/ftp_parameters.xml',
    'data/ir_cron.xml',
    'wizard/ftp_upload_views.xml',
],
```

```
id,name,model_id,group_id/id,operation,domain
access_ftp_upload_wzd,ej22.ftp.upload.wzd,ej22.ftp.upload.wzd,base.group_system,crud,
```

El `AbstractModel` no necesita permiso porque no tiene registros.

Un FTP de pruebas en tu ordenador

```
pip install pyftplib
mkdir -p /tmp/ftp/subidas
python -m pyftplib -p 2121 -u curso -P clave -d /tmp/ftp -w
```

Cambia `ej22_ftp.port` a 2121 y crea `ej22_ftp.password = clave`.

Prueba

1. Abre el asistente, sube un PDF y comprueba que aparece en `/tmp/ftp/subidas`.
2. Ajustes → Técnico → Tareas programadas → *Subir contactos al FTP* → *Ejecutar manualmente*. Aparece `contactos_AAAAMMDD.csv`.

Resultado en Odoo 20

La tarea programada:

Contatos Contatos Configuración

Nuevo Acciones planificadas Subir contatos al FTP 1/1

Ejecutar manualmente

Subir contatos al FTP

Modelo ? Servicio FTP

Usuario del planificador OdooBot

Ejecutar cada ? 1 Dias

Activo ☐

Siguiente fecha de ejecución ? 31 oct, 3:00

Prioridad ? 5

Código Ayuda Explanation

```
1 model.cron_upload_partners()
```

Enviar mensaje Registrar una nota Actividad

Hoy

OdooBot 12:41 Scheduled Action creado

El asistente de subida:

Contatos Contatos Configuración

Subir a FTP

Fichero Suba su archivo

Subir Cancelar

Y los ficheros en el servidor FTP de pruebas, con el CSV que genera el cron:

```
Terminal — servidor FTP de pruebas

$ ls -l /tmp/ftp/subidas
total 8
-rw-r--r-- 1 root root 468 Oct 30 12:44 contactos_20261030.csv
-rw-r--r-- 1 root root 22 Oct 30 12:44 saludo.txt
$ head -4 /tmp/ftp/subidas/contactos_20261030.csv
id,nombre,email,telefono
10,Acme Corporation,acme_corp@yourcompany.example.com,(603)-996-3829
16,Azure Interior,azure.Interior24@example.com,(870)-931-0505
11,Gemini Furniture,gemini_furniture@fake.geminifurniture.com,+378 0549 885555
```

Reto

Guarda la fecha de la última subida correcta en un parámetro `ej22_ftp.last_upload` con `set_str` y muéstrala en el asistente.

Capítulo 23 · Tu propia API REST con token

Crearemos una ruta `GET /api/partner/<NIF>` que devuelve los datos de un contacto en JSON, protegida con tokens que se gestionan desde Odoo. Ejercicio original: `18_api_controller`.

¿JSON-2 o controlador propio?

La API JSON-2 del capítulo 21 da acceso a **todo** lo que el usuario puede ver. Un controlador propio es mejor cuando quieres exponer **solo** una operación concreta, con un formato fijo, a alguien que no debe tener usuario en Odoo (una web pública, un socio).

Paso 1 · El modelo de tokens

```
import secrets

from odoo import fields, models

class ApiToken(models.Model):
    _name = 'api.token'
    _description = 'Token de API'

    name = fields.Char(string='Cliente de la API', required=True)
    token = fields.Char(
        string='Token', required=True, copy=False, readonly=True,
        default=lambda self: secrets.token_urlsafe(32))
    is_active = fields.Boolean(string='Activo', default=True)
    last_use = fields.Datetime(string='Último uso', readonly=True)

    _token_uniq = models.UniqueIndex('(token)', 'El token debe ser único.')
```

- `secrets.token_urlsafe(32)` genera un token aleatorio y seguro. El original obligaba a escribirlo a mano.
- `UniqueIndex` impide tokens repetidos.

Paso 2 · Preparar la respuesta en el modelo

```
from odoo import models

class ResPartner(models.Model):
    _inherit = 'res.partner'

    def _prepare_api_values(self):
        self.ensure_one()
        return {
            'id': self.id,
            'name': self.name,
            'vat': self.vat,
            'street': self.street,
            'zip': self.zip,
            'city': self.city,
            'country': self.country_id.code,
            'email': self.email,
            'website': self.website,
        }
```

La lógica vive en el modelo; el controlador solo traduce HTTP a Python y viceversa.

Paso 3 · El controlador

controllers/main.py:

```
from odoo import fields, http
from odoo.http import request

class PartnerApi(http.Controller):

    @http.route('/api/partner/<string:vat>', type='http', auth='public',
                methods=['GET'], csrf=False, cors='*', save_session=False)
    def partner_info(self, vat, **kwargs):
        token_value = request.httprequest.headers.get('X-API-Token')
        token = request.env['api.token'].sudo().search([
            ('token', '=', token_value), ('is_active', '=', True),
        ], limit=1) if token_value else None
        if not token:
            return request.make_json_response({'error': 'Token no válido'}, status=401)

        token.last_use = fields.Datetime.now()
        partner = request.env['res.partner'].sudo().search([('vat', '=', vat)], limit=1)
        if not partner:
            return request.make_json_response(
                {'error': f'No existe un contacto con NIF {vat}'}, status=404)
        return request.make_json_response(partner._prepare_api_values())
```

controllers/__init__.py: from . import main. Y en el __init__.py raíz: from . import controllers, models.

Pieza	Explicación
type='http'	Petición HTTP normal (las de tipo jsonrpc son para el cliente web de Odoo)
auth='public'	No exige sesión; nosotros comprobamos el token
sudo()	El usuario público no tiene permisos: actuamos como superusuario, solo tras validar el token
X-API-Token	El token va en una cabecera, no en la URL: las URL quedan grabadas en logs y proxies
make_json_response	Devuelve JSON con el Content-Type y el código HTTP correctos
save_session=False	No crea una sesión por cada llamada

Paso 4 · Vista y menú de tokens

```
<odoo>
<record id="api_token_list" model="ir.ui.view">
  <field name="name">api.token.list</field>
  <field name="model">api.token</field>
  <field name="arch" type="xml">
    <list editable="bottom">
      <field name="name"/>
      <field name="token" widget="CopyClipboardChar"/>
      <field name="is_active" widget="boolean_toggle"/>
      <field name="last_use"/>
    </list>
  </field>
</record>
<record id="action_api_token" model="ir.actions.act_window">
  <field name="name">Tokens de API</field>
  <field name="res_model">api.token</field>
  <field name="view_mode">list</field>
</record>
<menuitem id="menu_api_token" name="Tokens de API" parent="base.menu_administration"
  action="action_api_token" sequence="91"/>
</odoo>
```

Permiso (solo administradores):

```
id,name,model_id,group_id/id,operation,domain
access_api_token_admin,api.token admin,api.token,base.group_system,crud,
```

Manifiesto: 'depends': ['base'].

Paso 5 · Probar

```
curl -s http://localhost:8069/api/partner/ESB12345678 -H "X-API-Token: TU_TOKEN"
# {"id": 14, "name": "Mi Cliente SL", "vat": "ESB12345678", ...}

curl -s -i http://localhost:8069/api/partner/ESB12345678
# HTTP/1.1 401 UNAUTHORIZED {"error": "Token no válido"}
```

Si tu servidor tiene varias bases de datos, la ruta pública necesita saber cuál usar: arranca Odoo con `--db-filter=^curso20$` o un solo `-d`.

Prueba

1. Ajustes — *Tokens de API* — Nuevo, nombre `tienda`. El token se genera solo.
2. Pon un NIF a un contacto y llama a la API con curl.
3. Desactiva el token y repite: responde 401.

Resultado en Odoo 20

Los tokens, con el último uso registrado:

Ajustes

Opciones generales

Usuarios y compañías

Subir a FTP

Tokens de API

6

6

My Company (San Francisco)

Nuevo

Tokens de API

Buscar...

1-1/1

<div>☐</div> Cliente de la API	Token	Activo	Último uso
<div>☐</div> tienda online	mrZ8ykOBi3nJxW-mAKn5m4BUOb_ZKVHScwSqtvUdhPk <div></div>	<div></div>	30 oct, 12:44

La respuesta JSON de `/api/partner/ESB12345674` con el token en la cabecera:

```
Pretty-print ☐
{"id": 9, "name": "Wood Corner", "vat": "ESB12345674", "street": "1839 Arbor Way", "zip": "95380", "city": "Turlock", "country": "US",
"email": "wood.corner26@example.com", "website": "http://www.wood-corner.com"}
```

Reto

Guarda solo el `hash` del token (`hashlib.sha256`) y muestra el token en claro una única vez, al crearlo, como hace Odoo con sus claves API.

Capítulo 24 · Una página web que busca empleados

Una página en `/empleados/buscar` con un formulario: escribes el número de identificación y muestra nombre, puesto y foto del empleado. Aprenderás a servir HTML desde Odoo con plantillas QWeb. Ejercicio original: `37_search_employee`.

Qué cambia respecto al original

- El original usaba **Jinja2** con `PackageLoader('odoo.addons.37_search_employee', ...)`. Funciona, pero se salta el sistema de plantillas, traducciones y estilos de Odoo. Aquí usamos **QWeb**, el motor de Odoo, con `request.render`.
- El formulario envía un GET normal (`?identificacion=...`): no hace falta JavaScript.
- La foto se lee con la API de binarios de Odoo 20 (`.to_base64()` y `.mimetype`).
- La ruta exige usuario (`auth='user'`). Mostrar datos de empleados a cualquiera que adivine un número de identificación es un riesgo de privacidad; si necesitas que sea pública, limita mucho los datos que enseñas.

Paso 1 · El controlador

`controllers/main.py`:

```
from odoo import http
from odoo.http import request

class EmployeeSearch(http.Controller):

    @http.route('/empleados/buscar', type='http', auth='user', methods=['GET'])
    def search_employee(self, identificacion=None, **kwargs):
        values = {'identificacion': identificacion or '', 'employee': None, 'photo': None}
        if identificacion:
            employee = request.env['hr.employee'].search([
                ('identification_id', '=', identificacion.strip()),
            ], limit=1)
            values['employee'] = employee
            if employee.image_256:
                image = employee.image_256
                values['photo'] = f'data:{image.mimetype};base64,{image.to_base64()}'
        return request.render('ej24_search_employee.employee_search_page', values)
```

- Sin `sudo()`: el usuario conectado solo ve los empleados que sus permisos le dejan ver.
- `identification_id` vive en `hr.version` desde Odoo 19, pero `hr.employee` lo hereda por `_inherits`, así que la búsqueda funciona igual.

Paso 2 · La plantilla QWeb

`views/employee_search_templates.xml`:

```
<odoo>
  <template id="employee_search_page" name="Buscar empleado">
    <t t-call="web.frontend_layout">
      <div class="container py-5" style="max-width: 640px;">
        <h1 class="mb-2">Buscar empleado</h1>
        <p class="text-muted">Escribe el número de identificación.</p>

        <form method="get" action="/empleados/buscar" class="d-flex gap-2 mb-4">
          <input type="text" name="identificacion" class="form-control"
            placeholder="Ej.: 12345678Z" t-att-value="identificacion" required="required"/>
```

```

        <button type="submit" class="btn btn-primary">Buscar</button>
    </form>

    <t t-if="employee">
        <div class="card">
            <div class="card-body d-flex gap-3 align-items-center">
                

                <div>
                    <h4 class="mb-1" t-out="employee.name"/>
                    <div t-out="employee.job_title or ''"/>
                    <div class="text-muted" t-out="employee.department_id.name or ''"/>
                    <div class="small">Identificación: <t t-out="employee.identification_id"/>
</div>

                </div>
            </div>
        </div>
    </t>
    <div t-elif="identificacion" class="alert alert-warning">
        No se encontró ningún empleado con la identificación <strong t-out="identificacion"/>.
    </div>
</div>
</t>
</template>
</odoo>

```

- `web.frontend_layout` aporta la página HTML completa con Bootstrap 5 y los estilos de Odoo: no hace falta cargar Bootstrap desde un CDN como el original.
- `t-out` escapa el texto automáticamente, evitando inyección de HTML.
- `t-att-value` rellena el campo con lo buscado.

Paso 3 · Manifiesto

```

{
    'name': 'Ej24 Buscar empleado',
    'version': '20.0.1.0.0',
    'license': 'LGPL-3',
    'depends': ['hr'],
    'data': ['views/employee_search_templates.xml'],
}

```

No hay modelos nuevos, así que no hay `ir.access.csv`.

Prueba

1. Empleados → abre uno → pestaña *Información privada* → rellena *Nº de identificación* con 12345678Z.
2. Abre `http://localhost:8069/empleados/buscar`, escribe 12345678Z y pulsa *Buscar*.
3. Prueba un número inexistente para ver el aviso.

Resultado en Odoo 20

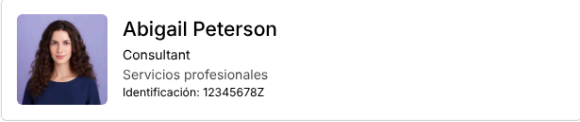
Búsqueda con un número existente:

 Mitchell Admin ▾

Buscar empleado

Escribe el número de identificación.

Buscar



Abigail Peterson
Consultant
Servicios profesionales
Identificación: 12345678Z

Copyright © My Company (San Francisco)

Con tecnología de 

Y con uno que no existe:

 Mitchell Admin ▾

Buscar empleado

Escribe el número de identificación.

Buscar

No se encontró ningún empleado con la identificación 00000000X.

Copyright © My Company (San Francisco)

Con tecnología de 

Reto

Añade al pie de la tarjeta un enlace *Abrir ficha* que lleve al formulario del empleado en el backend:
`/odoo/employees/<id>`.

Capítulo 25 · Odoo shell: Python directo contra tu base de datos

El *shell* de Odoo abre una consola Python con el entorno `env` ya cargado. Es la herramienta perfecta para probar código, consultar datos o corregir registros sin escribir un módulo. Ejercicio original: `33_shell_odoo`.

Paso 1 · Abrir el shell

```
# Instalación desde código
./odoo/odoo-bin shell -c odoo.conf -d curso20

# Docker (capítulo de instalación)
docker compose exec odoo odoo shell -d curso20 --db_host=db --db_user=odoo --db_password=odoo
```

Verás un prompt `>>>` con estas variables disponibles: `env`, `self` (el usuario administrador), `odoo` y `openerp`.

Paso 2 · Consultar

```
>>> partners = env['res.partner'].search([('active', '=', True)])
>>> len(partners)
52
>>> for p in partners[:5]:
...     print(p.id, p.name, p.email)

>>> env['res.partner'].search_count([('is_company', '=', True)])
>>> env['res.partner'].search_read([('country_id.code', '=', 'ES')], ['name', 'city'], limit=5)
>>> env['ej.pet'].search([]).mapped('name')
['Coco', 'Pinki', 'Rex']
```

Paso 3 · Modificar y guardar

En el shell **nada se guarda hasta que haces commit**:

```
>>> pet = env['ej.pet'].search([('name', '=', 'Coco')])
>>> pet.age = 3
>>> env.cr.commit()      # ahora sí queda guardado
```

Si te equivocas antes del commit: `env.cr.rollback()`.

Paso 4 · Borrar mensajes antiguos (del original, corregido)

```
>>> old = env['mail.message'].search([('date', '<', '2022-12-31')], limit=500)
>>> len(old)
>>> old.unlink()
>>> env.cr.commit()
```

El original tenía `limit=0`, un carácter extraño que provocaba un error de sintaxis. Usa siempre un `limit` numérico al borrar en lote y comprueba `len()` antes de `unlink()`.

Paso 5 · Ejecutar un script entero

Puedes pasar un fichero al shell por la entrada estándar:

```
./odoo/odoo-bin shell -c odoo.conf -d curso20 < arreglar_emails.py
```

arreglar_emails.py:

```
partners = env['res.partner'].search([('email', 'ilike', ' ')])
for partner in partners:
    partner.email = partner.email.replace(' ', '')
print(f'Corregidos {len(partners)} emails')
env.cr.commit()
```

Buenas prácticas

Haz	Evita
Probar primero en una copia de la base de datos	Ejecutar <code>unlink()</code> masivos en producción sin copia
<code>search_count</code> y <code>len()</code> antes de modificar	Commits a mitad de un bucle largo
<code>env['modelo'].with_context(lang='es_ES')</code> para ver traducciones	SQL directo (<code>env.cr.execute</code>) si el ORM puede hacerlo
Guardar los scripts útiles en tu repositorio	Teclear a mano la misma corrección cada mes

Prueba

Abre el shell, cuenta tus mascotas, cambia la edad de una, haz commit y comprueba el cambio en la interfaz.

Resultado en Odoo 20

Una sesión real del shell de Odoo 20:

```
Terminal — odoo-bin shell -d curso20

$ ./odoo/odoo-bin shell -c odoo.conf -d curso20
>>> partners = env['res.partner'].search([('active', '=', True)])
>>> len(partners)
39
>>> env['ej.pet'].search([]).mapped('name')
['Coco', 'Pinki', 'Rex']
>>> pet = env['ej.pet'].search([('name', '=', 'Coco')])
>>> pet.age
2
>>> pet.age = 3
>>> env.cr.commit()
>>> env['ej.pet'].search([('name', '=', 'Coco')]).age
3
```

Reto

Escribe un script que ponga `pretty_name = True` a todas las mascotas cuyo nombre tenga menos de 5 letras.

Parte VII · Odoo e inteligencia artificial

Capítulo 26 · Programar Odoo con Claude

Un asistente de IA como Claude puede escribir un módulo completo, explicarte código ajeno, migrar un módulo de Odoo 16 a 20 o escribir tests. No sustituye lo que has aprendido en este libro: lo multiplica. Quien entiende modelos, vistas y seguridad sabe pedir bien y, sobre todo, sabe **revisar** lo que la IA entrega.

Este libro es un ejemplo: los 23 módulos se reescribieron para Odoo 20 con ayuda de Claude, se instalaron en un Odoo 20 real y las capturas de cada capítulo salen de esa instalación.

Dos formas de trabajar

Herramienta	Dónde se usa	Ideal para
Claude (web, escritorio o móvil)	Conversación en el navegador o la app	Entender un concepto, revisar un fragmento, pedir un ejemplo
Claude Code	Terminal o editor, dentro de la carpeta de tu proyecto	Crear y modificar módulos completos, ejecutar Odoo y tests, corregir errores leyendo el log

La gran diferencia: Claude Code **trabaja sobre tus ficheros** y puede ejecutar comandos (con tu permiso), así que puede escribir el módulo, instalarlo, leer el error y corregirlo en bucle.

Paso 1 · Instalar Claude Code

```
# macOS y Linux
curl -fsSL https://claude.ai/install.sh | bash

# Windows (PowerShell)
irm https://claude.ai/install.ps1 | iex
```

Necesitas una suscripción de Claude (Pro o Max) o una clave de la API de Anthropic. La primera vez que lo arrancas te pide iniciar sesión.

Consulta la [documentación oficial de Claude Code](#) para el método de instalación más actual.

Paso 2 · Arrancarlo en tu proyecto

```
cd ~/curso-odoo20
claude
```

Claude Code lee la carpeta en la que lo abres. Ábrelo en la raíz donde están `odoo/` y `mis_modulos/` (o la carpeta `addons/` de Docker).

Paso 3 · Dale contexto con `CLAUDE.md`

`CLAUDE.md` es un fichero en la raíz del proyecto que Claude Code lee al empezar cada sesión. Es donde le cuentas las reglas de tu proyecto. Puedes generarlo con el comando `/init` y completarlo. Para Odoo 20, este es un buen punto de partida (resume la chuleta del anexo):

```
# Proyecto: módulos Odoo 20

## Entorno
- Odoo 20.0, Python 3.12+, PostgreSQL 16
- Código de Odoo en ./odoo, nuestros módulos en ./mis_modulos
- Base de pruebas: curso20
- Instalar/actualizar: ./odoo/odoo-bin -c odoo.conf -d curso20 -u MODULO --stop-after-init

## Reglas de Odoo 20 (obligatorias)
- Seguridad en security/ir.access.csv (id,name,model_id,group_id/id,operation,domain). Nunca ir.model.access.csv ni ir.rule.
- Toda fila de ir.access lleva grupo, salvo que se quiera una restricción global.
- Vistas con <list>, nunca <tree>. Nada de attrs ni states: invisible="expr".
- create con @api.model_create_multi y vals_list.
- Command.create/link/set en lugar de tuplas (0, 0, vals).
- Restricciones con models.Constraint / models.UniqueIndex.
- Mensajes traducibles con self.env._().
- Parámetros con get_bool/get_int/get_str.
- Binarios: .content y BinaryBytes; adjuntos con 'raw'.
- Integraciones nuevas con la API JSON-2, no XML-RPC.

## Estilo
- Nombres técnicos de módulo en minúsculas: ej27_nombre
- Etiquetas en español; código y comentarios claros para principiantes
- Después de cada cambio: instalar el módulo y revisar el log sin ERROR
```

Sin este fichero, una IA tiende a escribir código de versiones antiguas (hay muchos más ejemplos de Odoo 14–16 en internet que de Odoo 20). Con él, sigue tus reglas desde el primer mensaje.

Paso 4 · Pedir un módulo

Un buen encargo dice **qué, dónde y cómo comprobarlo**:

```
Crea el módulo ej27_vet_visits en mis_modulos:
- Modelo vet.visit con mascota (Many2One a ej.pet), fecha, motivo,
  importe y estado (borrador, hecha, cancelada).
- Vista lista, formulario con barra de estado y botones para cambiar
  de estado, acción y menú bajo Mascotas.
- Botón inteligente en la mascota con el número de visitas.
- Permisos para base.group_user.
Cuando termines, instálalo en curso20 y corrige cualquier error del log.
```

Claude Code planificará los ficheros, los escribirá, ejecutará el comando de instalación y leerá el log. Te pedirá permiso antes de ejecutar comandos o modificar ficheros; lee lo que propone antes de aceptar.

Paso 5 · Pedir tests

Odoo trae un sistema de tests. Pide uno para cada regla importante:

```
Escribe tests para ej27_vet_visits que comprueben que una visita
hecha no puede volver a borrador y que el contador de la mascota
se actualiza. Ejecútalos.
```

El resultado será algo así, en tests/test_vet_visit.py:

```
from odoo.exceptions import UserError
from odoo.tests import TransactionCase

class TestVetVisit(TransactionCase):

    def test_visit_count(self):
```

```

pet = self.env['ej.pet'].create({'name': 'Coco'})
self.env['vet.visit'].create({'pet_id': pet.id, 'reason': 'Vacuna'})
self.assertEqual(pet.visit_count, 1)

def test_done_cannot_go_back(self):
    pet = self.env['ej.pet'].create({'name': 'Rex'})
    visit = self.env['vet.visit'].create({'pet_id': pet.id, 'reason': 'Revisión'})
    visit.action_done()
    with self.assertRaises(UserError):
        visit.action_draft()

```

Y se ejecutan con:

```

./odoo/odoo-bin -c odoo.conf -d curso_tests -i ej27_vet_visits \
--test-enable --test-tags /ej27_vet_visits --stop-after-init

```

Los tests son tu red de seguridad: cuando la IA (o tú) cambie algo mañana, sabrás al momento si se ha roto.

Paso 6 · Migrar un módulo antiguo

Este es el uso estrella para quien tiene módulos de Odoo 16:

```

Migra mis_modulos/viejo_modulo de Odoo 16 a Odoo 20 siguiendo
CLAUDE.md. Primero ejecuta odoo-bin upgrade_code, después corrige
lo que falte (seguridad a ir.access.csv, tree a list, attrs...),
instálalo y explicame cada cambio.

```

Pedir que explique los cambios convierte la migración en una lección.

Paso 7 · Usarlo para aprender

También puedes usar Claude como profesor particular de este libro:

- «Explicame línea a línea el `create` del capítulo 6».
- «¿Por qué en el capítulo 8 el contador de teléfonos no lleva `store=True`?».
- «Ponme un ejercicio parecido al capítulo 13 pero con contactos y corrígemelo».
- «Tengo este error en el log: ... ¿qué significa?».

Lo que no debes delegar

Riesgo	Qué hacer
Código que "parece" correcto pero es de otra versión	Revisarlo contra la chuleta del anexo e instalarlo siempre
Permisos demasiado abiertos (<code>sudo()</code> , filas sin grupo)	Revisar a mano <code>ir.access.csv</code> y cada <code>sudo()</code>
Lógica contable o fiscal	Validarla con alguien de contabilidad; probar en una copia
Datos reales de clientes	Trabajar con una base de pruebas, nunca con producción
Contraseñas y claves API	No pegarlas en el chat ni en <code>CLAUDE.md</code> ; usar variables de entorno

Prueba

Instala Claude Code, crea `CLAUDE.md` con la plantilla de este capítulo y pide el módulo `ej27_vet_visits`. Revisa cada fichero que genere y compáralo con lo que has aprendido: deberías reconocer cada pieza.

Reto final

Pide a Claude que añada al módulo de visitas un informe PDF (capítulo 11), una exportación a Excel (capítulo 16) y una ruta API con token (capítulo 23). Después, explícale tú a Claude por qué cada parte está hecha así. Si puedes explicarlo, has terminado el curso.

Anexo · Chuleta de migración de Odoo 16 a Odoo 20

Todos los cambios que han aparecido en el libro, en una sola tabla. Úsala al pasar cualquier módulo antiguo a Odoo 20.

Manifiesto y entorno

Antes (16)	Odoo 20	Capítulo
'version': '16.0.1.0.0'	'version': '20.0.1.0.0'	1
Python 3.8–3.11	Python 3.12–3.14	Introducción
PostgreSQL 12+	PostgreSQL 16+	Introducción
Demo cargada por defecto	--with-demo para cargarla	5
Raíz XML <openerp>	Siempre <odoo>	17

Seguridad

Antes (16)	Odoo 20	Capítulo
security/ir.model.access.csv	security/ir.access.csv	1
Columnas model_id:id,group_id:id,perm_read,...	model_id,group_id/id,operation,domain	1
model_ej_pet en model_id	Nombre técnico ej.pet	1
Permisos 1,1,1,1	Letras crud (c crear, r leer, u modificar, d borrar)	1
Fila sin grupo = acceso para todos	Fila sin grupo = restricción para todos	1
ir.rule con domain_force	Columna domain en ir.access.csv	8
res.groups campo users	user_ids	8,9
res.groups campo category_id	privilege_id – res.groups.privilege	8,9

ORM y Python

Antes (16)	Odoo 20	Capítulo
@api.model + create(self, vals)	@api.model_create_multi + create(self, vals_list)	6
super(Clase, self).metodo()	super().metodo()	6
(0, 0, vals), (4, id), (6, 0, ids)	Command.create(vals), Command.link(id), Command.set(ids)	13,18
_sql_constraints = [...]	models.Constraint, models.UniqueIndex, models.Index	8
('texto')	self.env.('texto')	4
env.ref(xmlid).read()[0]	env['ir.actions.act_window']._for_xml_id(xmlid)	8,14
get_param('clave') – texto	get_bool, get_int, get_float, get_str	15
get_values / set_values en ajustes	Atributo config_parameter= en el campo	15
Binario = texto base64	Objeto BinaryValue: .content, .to_base64(), .mimetype	17,24
Adjunto con datas (base64)	Adjunto con raw (bytes)	16
Dominios como listas sumadas	Clase Domain combinable con & y	10
move.post()	move.action_post()	18,19
Pago: payment_method_id, ref	payment_method_line_id, memo; mejor account.payment.register	20
Producto type='product'	type='consu' + is_storable=True	16,17

Antes (16)	Odoo 20	Capítulo
identification_id en hr.employee	En hr.version, heredado por _inherits	24
Cron con numbercall, doall	Ya no existen	22
@route(type='json')	@route(type='jsonrpc')	23

Vistas e informes

Antes (16)	Odoo 20	Capítulo
<tree>	<list>	1
view_mode = tree, form	list, form	1
attrs="{ 'invisible': [('x', '=', True)] }"	invisible="x"	2
states="draft"	invisible="state != 'draft'"	20
<div class="oe_chatter">...	<chatter/>	7
Iconos fa-phone	Material Symbols: call, receipt_long	8
Ajustes con o_setting_box	<app>, <block>, <setting>	15
Etiqueta <report .../>	Registro ir.actions.report	11
t-esc, t-raw	t-out (o t-field)	7, 11
Bootstrap 4: text-right, ml-1	Bootstrap 5: text-end, ms-1	11
Jinja2 en controladores	QWeb con request.render	24

Integraciones

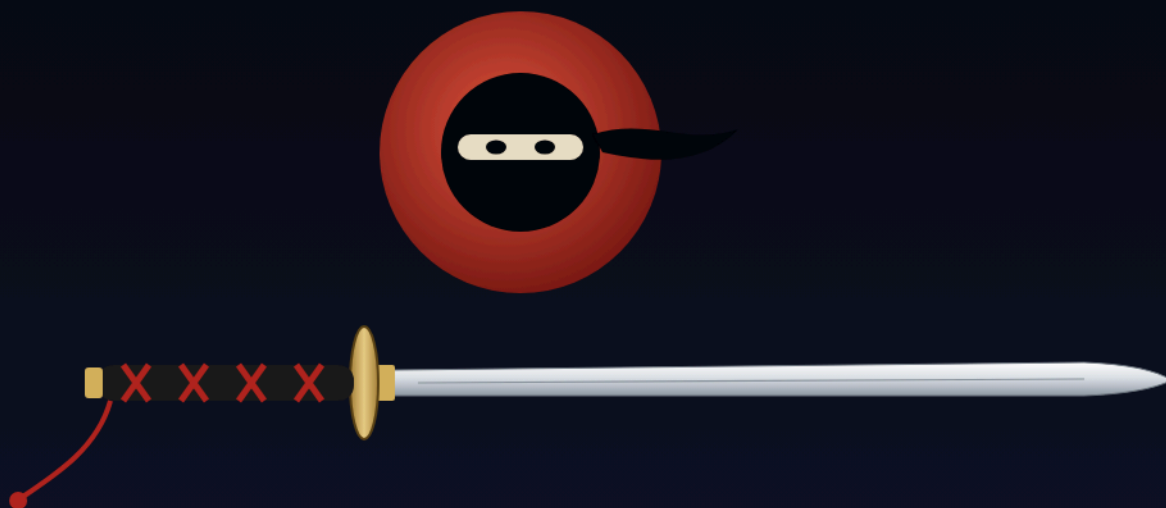
Antes (16)	Odoo 20	Capítulo
/xmlrpc/2/object con contraseña	/json/2/<modelo>/<método> con clave API	21
execute(db, uid, pwd, modelo, método, args)	JSON con argumentos por nombre	21
XML-RPC sin fecha de fin	Obsoleto desde 19, se elimina en Odoo 22	21
xlwt (.xls)	XlsxWriter (.xlsx)	16
xlrd para .xlsx	openpyxl	17

Herramienta automática

Odoo trae un asistente que aplica muchos de estos cambios por ti. Ejecútalo sobre una copia de tu código, con Git limpio, y revisa el resultado:

```
./odoo/odoo-bin upgrade_code --addons-path=mis_modulos
git diff
```

No sustituye a la revisión manual: la seguridad (ir.access.csv) y la lógica contable hay que comprobarlas a mano, como has visto en este libro.



Aprende Odoo 20 como un ninja: en silencio, paso a paso y sin miedo.

Veintiséis capítulos que llevan a un principiante desde su primer módulo hasta facturas, pagos, APIs e inteligencia artificial.

Cada ejercicio se ha probado en un Odoo 20 real.

- ◆ Instalación en Windows, Mac y Docker
- ◆ Python, arquitectura y primer módulo
- ◆ Vistas, herencia, permisos con ir.access
- ◆ Informes PDF y Excel, asistentes, ajustes
- ◆ Facturas, asientos, pagos y conciliación
- ◆ API JSON-2, controladores, FTP y cron
- ◆ Programar Odoo con Claude

Marlon Falcón Hernández

Especialista en Odoo ERP · Implemento BIM 2.0 · Valencia, España

marlonfalcon.com · github.com/falconsoft3d

