

### Caso práctico



[pexels](#) (CC0)

En **Datalab** ya llevan más de tres meses funcionando con su nuevo ERP. El proceso de implantación ha sido largo y complicado, pero han ganado en funcionalidad y productividad. Con el nuevo sistema es posible trabajar mucho más rápido y mejor.

**Datalab** ha contratado un servicio de mantenimiento con **BK Programación**. Este servicio incluye la realización de unos módulos nuevos que necesita la empresa. Los responsables de **Datalab** se han puesto en contacto con **Ada** para interesarse sobre cómo van los avances de esos módulos a medida. **Ada** les ha dicho que una vez terminado el proceso de implantación inicial, ya pueden empezar a desarrollar los módulos a medida.

En **BK Programación** van a desarrollar los nuevos módulos bajo el paradigma de Modelo-Vista-Controlador. Este modelo lo han podido ver frecuentemente en aplicaciones web donde:

- ✓ La vista es la página web y el código que proporciona de datos dinámicos a la página.
- ✓ El modelo es el Sistema de Gestión de Base de Datos.
- ✓ El controlador es el responsable de recibir las peticiones del usuario a través de la vista, consultar datos del modelo, realizar los cálculos necesarios y solicitar nuevas vistas.



[Ministerio de Educación y Formación Profesional](#). (Dominio público)

**Materiales formativos de FP Online propiedad del Ministerio de Educación y Formación Profesional.**

[Aviso Legal](#)

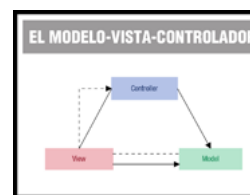
# 1.- Técnicas y estándares. Modelo-Vista-Controlador.

¿Utilizas habitualmente hojas de cálculo, como por ejemplo Calc de OpenOffice.org? En ese caso, sabrás que al introducir datos en las celdas, los mismos datos podemos verlos de varias formas. Es posible ver los datos en la hoja de cálculo donde los hemos introducido, o mediante un gráfico que puede ser de barras, circular, etc.

Esos programas implementan el patrón Modelo-Vista-Controlador (MVC), el modelo lo constituyen los datos que hemos introducido en las celdas. Las vistas se encargan de mostrar los datos de la forma que se seleccione.

El MVC divide una aplicación en tres componentes:

- ✓ Los datos de la aplicación (**modelo**).
- ✓ La interfaz del usuario (**vista**).
- ✓ El **controlador**, el cual define la forma en que la interfaz reacciona a la entrada del usuario.



Elaboración propia (Uso educativo no comercial)

Con esto se consigue separar los datos (modelo) de la interfaz de usuario (vista), de manera que los cambios en la interfaz no afectan a los datos y viceversa, es decir, que los datos pueden ser cambiados sin afectar a la interfaz de usuario.

En ODOO, el MVC se implementa de la siguiente forma:

- ✓ El **modelo** son las tablas de la base de datos.
- ✓ La **vista** son los archivos XML que definen la interfaz de usuario del módulo.
- ✓ El **controlador** son los objetos creados en Python.

Por ejemplo, en el diagrama anterior, las flechas continuas significan que el controlador tiene un acceso completo a la vista y al modelo, y las flechas discontinuas significan un acceso limitado. Las razones de este diseño son las siguientes:

- ✓ Acceso de **modelo** a **vista**: el modelo envía una notificación a la vista cuando sus datos han sido modificados, con el fin de que la vista pueda actualizar su contenido. El modelo no necesita conocer el funcionamiento interno de la vista para realizar esta operación. Sin embargo, la vista necesita acceder a las partes internas del controlador.
- ✓ Acceso de **vista** a **controlador**: la razón de que la vista tenga limitado el acceso al controlador es porque las dependencias que la vista tiene sobre el controlador deben ser mínimas ya que el controlador puede ser sustituido en cualquier momento.

## Autoevaluación

**En el modelo-vista-controlador la vista son los archivos XML, el modelo las tablas y el controlador el código que define el comportamiento de la aplicación.**

☐ Verdadero ☐ Falso

**Verdadero**

Es la separación entre los componentes de la aplicación que establece el modelo-vista-controlador.

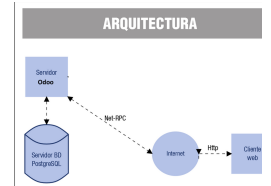
## 1.1.- Especificaciones técnicas para el desarrollo de componentes.

¿Recuerdas en la primera unidad cuando hablábamos de la arquitectura de los sistemas de planificación empresarial? Decíamos que la mayoría están basados en una arquitectura cliente-servidor, este es el caso de Odoo.

Odoo utiliza los protocolos XML-RPC o Net-RPC para comunicación entre cliente y servidor. Básicamente lo que hacen es permitir al cliente hacer llamadas a procedimientos remotos, o sea, ejecutar código de otra máquina. En el caso de XML-RPC, la función llamada, sus argumentos y el resultado se envían por HTTP y son codificadas usando XML. Net-RPC está disponible más recientemente, está basado en funciones en Python y es más rápido.

ODOO funciona sobre un marco de trabajo o framework llamado OpenObject. Este framework permite el desarrollo rápido de aplicaciones (RAD), siendo sus principales elementos los siguientes:

- ✓ ORM: mapeo de bases de datos relacionales a objetos Python.
- ✓ Arquitectura MVC (Modelo Vista Controlador).
- ✓ Diseñador de informes.
- ✓ Herramientas de Business Intelligence y cubos multidimensionales.
- ✓ Cliente web.



Elaboración propia (Uso educativo no comercial)

### Para saber más

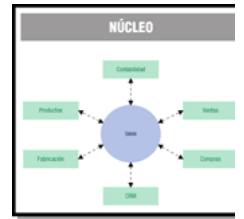
Para mayor información sobre el protocolo XML-RPC puedes consultar el siguiente enlace:

[Información sobre el protocolo XML-RPC.](#)

## 1.2.- Especificaciones funcionales para el desarrollo de componentes.

Como sabemos, Odoo se compone de un núcleo y varios módulos dependiendo de las necesidades, por ejemplo:

- ✔ **base:** es el módulo básico compuesto de objetos como empresas (`res.partner`), direcciones de empresas (`res.partner.address`), usuarios (`res.user`), monedas (`res.currency`), etc.
- ✔ **account:** gestión contable y financiera.
- ✔ **product:** productos y tarifas.
- ✔ **purchase:** gestión de compras.
- ✔ **sale:** gestión de ventas.
- ✔ **mrp:** fabricación y planificación de recursos.
- ✔ **crm:** gestión de las relaciones con clientes y proveedores.



Elaboración propia (Uso educativo no comercial)

Por cada módulo existe una carpeta con el nombre del módulo en el directorio `addons` del servidor. Para crear un módulo tendremos que crear una carpeta dentro de ese directorio, y seguir los siguientes pasos:

1. Crear el archivo de inicio de módulo: `__init__.py`
2. Crear el archivo con la descripción del módulo: `__manifest__.py`.
3. Crear los archivos Python con la definición de objetos: `nombre_modulo.py`, por medio de clases del lenguaje Python definiremos el modelo y el controlador.
4. Vista o vistas del objeto `nombre_modulo_nombre_objeto.xml`.

En la carpeta `addons` además de los archivos anteriores, puede haber otras subcarpetas como `report`, `wizard`, `test`, que contienen archivos del tipo de objeto utilizado, en este caso, informes, asistentes y accesos directos, datos de prueba, etc.

El archivo `__manifest__.py` debe contener los siguientes valores dentro de un diccionario Python (estructura de datos cerrada con llaves `{ }`):

- ✔ **name:** nombre del módulo.
- ✔ **version:** versión del módulo.
- ✔ **description:** una descripción del módulo.
- ✔ **author:** persona o entidad que ha desarrollado el módulo.
- ✔ **website:** sitio web del módulo.
- ✔ **license:** tipo de licencia del módulo (por defecto `GPL`).
- ✔ **depends:** lista de módulos de los que depende el módulo, el módulo base es el más usado ya que en él se definen algunos datos que son necesarios las vistas, informes, etc.
- ✔ **init\_xml:** lista de los archivos XML que se cargarán con la instalación del módulo.
- ✔ **installable:** determina si el módulo es instalable o no.

### Para saber más

Para mayor información sobre XML puedes consultar el siguiente enlace:

[Introducción a XML.](#)

### Autoevaluación

Un módulo en Odoo está formado por los siguientes archivos: `__init__.py`, `__manifest__.py`, `nombre_modulo.py` y `nombre_modulo_nombre_objeto.xml`.

☐ Verdadero ☐ Falso

Verdadero

En las versiones anteriores el archivo `__manifest__.py`, se llamaba `__terp__.py` o `__openerp__.py`

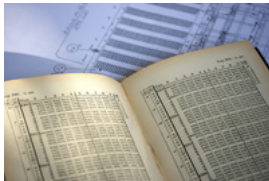
## 2.- Técnicas de optimización de consultas y acceso a grandes volúmenes de información.

### Caso práctico

**María** está realizando un mantenimiento de la base de datos. Ha observado que determinadas consultas tienen un tiempo de respuesta algo menos rápido que otras y está analizando el código de las mismas. Se ha dado cuenta que hay consultas que no siguen las reglas de optimización, y probablemente ese sea el motivo de que necesiten más tiempo para extraer los datos de la aplicación.



Elaboración propia (Uso educativo no comercial)



Marcin Wichary (CC BY)

Un aspecto a tener en cuenta a la hora de hacer modificaciones en la aplicación y crear módulos que utilicen nuevos objetos, es que estos objetos se basan en consultas a la información existente en la base de datos. Si deseamos mejorar los tiempos de respuesta del sistema, deberemos crear objetos que utilicen consultas lo más optimizadas posibles. Podemos decir que la **optimización** es el proceso de modificar un sistema para mejorar su eficiencia o el uso de los recursos disponibles.

Cuando manejamos grandes cantidades de datos, el resultado de una consulta puede tomar un tiempo considerable, obteniendo no siempre una respuesta óptima. Dentro de las técnicas de optimización de consultas podemos encontrar las siguientes:

- ✓ **Diseño de tablas.** A la hora de crear nuevas tablas, asegurarnos de que no hay duplicidad de datos y que se aprovecha al máximo el almacenamiento en las tablas.
- ✓ **Campos.** Es recomendable ajustar al máximo el espacio en los campos para no desperdiciar espacio.
- ✓ **Índices.** Permiten búsquedas a una velocidad notablemente superior, pero no debemos estar incitados a indexar todos los campos de una tabla, ya que los índices ocupan más espacio y se tarda más al actualizar los datos. Dos de las razones principales para utilizar un índice son:
  - Es un campo utilizado como criterio de búsquedas.
  - Es una clave ajena en otra tabla.
- ✓ **Optimizar sentencias SQL.** Existen una serie de reglas para utilizar el lenguaje de consulta y modificación de datos que hay que contemplar. Estas reglas se refieren tanto a la utilización de las sentencias de selección, como a las que realizan alguna inserción o modificación en la base de datos.
- ✓ **Optimizar la base de datos.** También podemos conectarnos en modo comando a la base de datos y utilizar sentencias para optimizar los datos contenidos en la base de datos.

## 2.1.- Operaciones de consulta. Herramientas.

Para optimizar la base de datos necesitamos conectarnos con PostgreSQL en modo comando, los pasos serían los siguientes:

1.- Cambiarnos al usuario `postgres`. Esto debemos hacerlo porque tenemos que entrar en el monitor interactivo con un usuario que exista en PostgreSQL:

```
$ sudo su postgres
```

2.- Entramos en el monitor interactivo de PostgreSQL llamado `psql`:

```
$ psql
postgres@ubuntu:/home/profesor$ psql
psql (8.4.8)
Digite «help» para obtener ayuda.
postgres=#
```

3.- Una vez dentro del monitor interactivo, el `postgres=#` significa que el monitor está listo y que podemos escribir comandos.

4.- Salir del monitor de PostgreSQL, con el comando `\q`.

Entre los comandos que podemos utilizar tenemos:

- ✓ `\h` ----> Ayuda.
- ✓ `\l` ----> Muestra las bases de datos existentes.
- ✓ `\c [nombre_bd]` ----> Nos conectamos con la base de datos que queramos.
- ✓ `\d` ----> Muestra las tablas existentes en la base de datos.
- ✓ `\d [nombre_tabla]` ----> Muestra la descripción de una tabla, o sea, los campos que contiene, de qué tipo son, etc.
- ✓ `VACUUM VERBOSE ANALYZE [tabla];` ----> Limpia y analiza bases de datos.
- ✓ `\q` ----> Salimos del editor de consultas.



[Mecamático](#) (CC BY-SA)

### Debes conocer

En el siguiente enlace puedes ver una presentación sobre cómo conectar con el monitor interactivo de PostgreSQL:

◀ 1 2 3 4 5 6 7 ▶

### Monitor interactivo de PostgreSQL

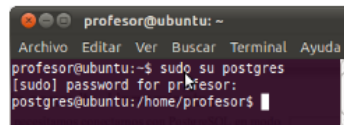
Elaboración propia (Uso educativo no comercial)

# Cómo entrar en el Monitor Interactivo de PostgreSQL

En esta presentación vamos a ver los pasos para entrar en el monitor interactivo de PostgreSQL e introducir comandos.

Elaboración propia (Uso educativo no comercial)

## Cambiar usuario

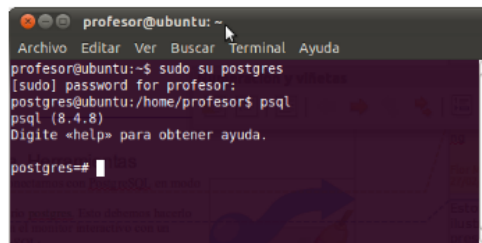


```
profesor@ubuntu: ~  
Archivo Editar Ver Buscar Terminal Ayuda  
profesor@ubuntu:~$ sudo su postgres  
[sudo] password for profesor:  
postgres@ubuntu:/home/profesor$
```

- Cambiamos al usuario postgres
- Esto debemos hacerlo porque tenemos que entrar en el monitor interactivo con un usuario que exista en la base de datos

Captura de pantalla de terminal de Ubuntu. ([GNU/GPL](#))

# Entrar

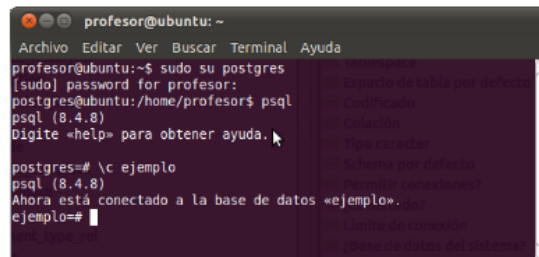


```
profesor@ubuntu: ~  
Archivo Editar Ver Buscar Terminal Ayuda  
profesor@ubuntu:~$ sudo su postgres  
[sudo] password for profesor:  
postgres@ubuntu:/home/profesor$ psql  
psql (8.4.8)  
Digite «help» para obtener ayuda.  
postgres=#
```

- Entramos en el monitor interactivo con la orden `psql`  
El prompt `postgres=#` significa que el sistema está listo para introducir comandos

Captura de pantalla de terminal de Ubuntu. (GNU/GPL)

# Introducir comandos I



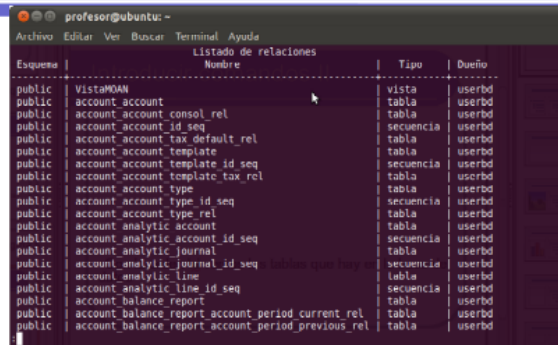
```
profesor@ubuntu: ~  
Archivo Editar Ver Buscar Terminal Ayuda  
profesor@ubuntu:~$ sudo su postgres  
[sudo] password for profesor:  
postgres@ubuntu:/home/profesor$ psql  
psql (8.4.8)  
Digite «help» para obtener ayuda.  
postgres=# \c ejemplo  
psql (8.4.8)  
Ahora está conectado a la base de datos «ejemplo».  
ejemplo=#
```

- Con la orden `\c [nombre_bd]` podemos conectarnos con una base de datos

Captura de pantalla de terminal de Ubuntu. (GNU/GPL)



## Introducir comandos II

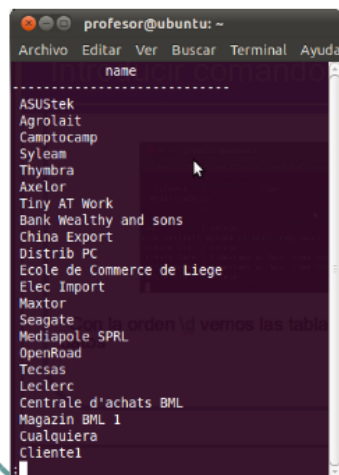


Esquema	Nombre	Tipo	Dueño
public	VistaMOAN	vista	userbd
public	account.account	tabla	userbd
public	account.account_consol_rel	tabla	userbd
public	account.account_id_seq	secuencia	userbd
public	account.account_tax_default_rel	tabla	userbd
public	account.account_template	tabla	userbd
public	account.account_template_id_seq	secuencia	userbd
public	account.account_template_tax_rel	tabla	userbd
public	account.account_type	tabla	userbd
public	account.account_type_id_seq	secuencia	userbd
public	account.account_type_rel	tabla	userbd
public	account.analytic.account	tabla	userbd
public	account.analytic.account_id_seq	secuencia	userbd
public	account.analytic.journal	tabla	userbd
public	account.analytic.journal_id_seq	secuencia	userbd
public	account.analytic.line	tabla	userbd
public	account.analytic.line_id_seq	secuencia	userbd
public	account.balance.report	tabla	userbd
public	account.balance.report.account_period_current_rel	tabla	userbd
public	account.balance.report.account_period_previous_rel	tabla	userbd

- Con la orden `\d` vemos las tablas que hay en la base de datos

Captura de pantalla de terminal de Ubuntu. (GNU/GPL)

## Introducir comandos III



name
ASUSTek
Agrolait
Camptocamp
Syleam
Thymbra
Axelor
Tiny AT Work
Bank Wealthy and sons
China Export
Distrib PC
Ecole de Commerce de Liege
Elec Import
Maxtor
Seagate
Mediapole SPRL
OpenRoad
Tecsas
Leclerc
Centrale d'achats BML
Magasin BML 1
Cualquiera
Cliente1

- Para trabajar con las tablas introducimos órdenes SQL, por ejemplo para ver los nombres de las empresas introducimos la orden:

`select name from res_partner;`

- Para salir usamos la orden `\q`

Captura de pantalla de terminal de Ubuntu. (GNU/GPL)

## Para saber más

En el siguiente enlace puedes encontrar un tutorial de PostgreSQL que contiene información sobre el monitor interactivo `psql` :

[Tutorial sobre PostgreSQL.](#)

## 2.2.- Sistemas batch inputs. Generación de programas de extracción y procesamiento de datos.

El término batch-input procede de la utilización en sistemas SAP de un método utilizado para transferir grandes cantidades de datos a un sistema ERP. Existen dos formas de hacer un batch-input:

- ✓ **Método clásico.** Método asíncrono, es decir, que se procesan los datos pero se actualizan más tarde. Tiene la característica de que genera un archivo de mensajes o log para tratarse errores a posteriori.
- ✓ **Método "call transaction".** Método on-line usado para dar de alta rápidamente pocos registros. Es un método síncrono, no genera log es mucho más rápido pero poco útil para gran cantidad de datos.



[Marc. Smith \(CC BY\)](#)

Un proceso batch-input se compone de dos fases:

- ✓ **Fase de generación.** Es la fase en que se genera el archivo batch-input con los datos a introducir o modificar.
- ✓ **Fase de procesamiento.** El archivo batch-input se ejecuta, haciéndose efectivas las modificaciones en la base de datos.

### Autoevaluación

**El término batch-input procede del procesamiento de grandes cantidades de información en los sistemas SAP.**

☐ Verdadero ☐ Falso

**Verdadero**

Requieren programas tanto par la generación del archivo como para el procesamiento posterior.

### 3.- Lenguaje proporcionado por los sistemas ERP-CRM.

#### Caso práctico



Elaboración propia (Uso educativo no comercial)

**María** le ha pasado a **Juan** las especificaciones de los nuevos módulos a realizar a Datalab. **Juan** le está explicando que el lenguaje de programación a utilizar es Python.

—Es un lenguaje que se suele utilizar para aprender a programar, por su sencillez —comenta **Juan**—. Sus instrucciones son muy parecidas al lenguaje natural.

—Sí —responde **María**—, eso había observado.

—Si te parece yo me encargo de programar los módulos, lo que sería hacer el modelo y el controlador.

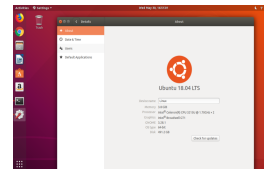
—De acuerdo -asiente **María**-, así puedo centrarme en los archivos XML para hacer la vista de los módulos. Hay que tener en cuenta que hay que hacer dos tipos de vista para cada módulo: una de tipo formulario y otra de tipo árbol, para facilitar las búsquedas.

—Muy bien -responde **Juan**—, pues ¡manos a la obra!.

La mayoría de aplicaciones de planificación empresarial están construidas con lenguajes de programación modernos y, en muchos casos, orientados a objetos. Existen aplicaciones ERP que proporcionan lenguajes propietarios, que no son lenguajes estándar y requieren que el programador se forme en ese lenguaje sólo y exclusivamente para manejar el ERP.

El lenguaje de programación utilizado en Odoo es el lenguaje Python.

Este lenguaje fue creado por Guido van Rossum a principios de los años 90 en el Centro para las Matemáticas y la Informática (CWI, Centrum Wiskunde & Informatica), en los Países Bajos. Guido sigue siendo el autor principal de Python.



Mrsinghpammar (CC BY-SA)

#### ¿Sabías qué?

El nombre del lenguaje Python proviene de la afición de su creador original por el grupo de cómicos ingleses "Monty Python".

En 2001, se creó la Python Software Foundation (PSF), una organización sin fines de lucro creada específicamente para poseer la propiedad intelectual sobre la licencia y promover el uso del lenguaje. Como miembros patrocinadores de esta organización están Google, Canonical y Microsoft.

La licencia de Python es de código abierto (Python Software Foundation License, PSFL), compatible con GPL. Es un lenguaje muy sencillo de utilizar con una sintaxis muy cercana al lenguaje natural, por lo que se considera uno de los mejores lenguajes para empezar a programar.

Python es utilizado por Google, Yahoo, la NASA o por ejemplo en todas las distribuciones Linux, donde los programas escritos en este lenguaje representan un porcentaje cada vez mayor. Aunque ver en profundidad este lenguaje nos llevaría más tiempo del esperado, vamos a ver algunas de sus características más importantes para poder crear nuestros propios módulos en Odoo.

#### Para saber más

Si deseas conocer más sobre el lenguaje Python puedes acceder a su web oficial:

[Web oficial del lenguaje Python.](http://www.python.org)

## 3.1.- Características y sintaxis del lenguaje.



[Docsearls \(CC BY-SA\)](#)

Las principales características de Python son:

- ✔ **Sintaxis muy sencilla**, lo cual facilita un rápido aprendizaje del lenguaje.
- ✔ **Lenguaje interpretado**. Los programas en Python se ejecutan utilizando un programa intermedio llamado intérprete. El código fuente se traduce a un archivo llamado `bytecode` con extensión `.pyc` o `.pyo`, que es el que se ejecutará en sucesivas ocasiones.
- ✔ **Tipado dinámico**. Esta característica quiere decir que los datos no se declaran antes de utilizarlos, sino que el tipo de una variable se determina en tiempo de ejecución, según el dato que se le asigne.
- ✔ **Fuertemente tipado**. Cuando una variable es de un tipo concreto, no se puede usar como si fuera de otro tipo, a no ser que se haga una conversión de tipo. Si la variable es numérica entera, por ejemplo, no podrá asignarle un valor real. En otros lenguajes, el tipo de la variable cambiaría para adaptarse al comportamiento esperado, aunque ello puede dar más lugar a errores.
- ✔ **Multiplataforma**. El intérprete de Python está disponible para una gran variedad de plataformas: Linux, Windows, Macintosh, etc.
- ✔ **Orientado a objetos**. Los programas están formados por clases y objetos, que son representaciones del problema en el mundo real. Los datos, junto con las funciones que los manipulan, son parte interna de los objetos y no están accesibles al resto de los objetos. Por tanto, los cambios en los datos de un objeto sólo afectan a las funciones definidas para ese objeto, pero no al resto de la aplicación.

Para hacer el primer programa en Python no necesitamos conocer mucho sobre el lenguaje, por ejemplo, podemos hacer el famoso "Hola mundo" con el que se suele comenzar a estudiar casi todos los lenguajes, simplemente con la siguiente línea:

```
print('Hola mundo')
```

Si escribimos esto en el intérprete de Python se ejecutará y se imprimirá por pantalla el mensaje "Hola mundo". ¡Ya tenemos nuestro primer programa! Probémoslo. El intérprete de Python está instalado por defecto en la mayoría de las distribuciones de Linux, para saber si lo tenemos instalado simplemente tenemos que introducir `python` en un terminal, y nos aparecerá un mensaje con la versión que tenemos instalada y el prompt `>>>`, que indica que está esperando código del usuario. Podemos salir escribiendo `exit()`, o con la combinación de teclas **Control + D**.

```
usu@:~$ python
Python 2.7.12 (default, Jul 21 2016, 15:19:50)
Type "help", "copyright", "credits" or "license()" for more information.
>>> print('Hola mundo')
Hola mundo
usu@:~$
```

Captura de pantalla de terminal de Ubuntu. ([GNU/GPL](#))

También podemos guardar el código en un archivo llamado `programa.py`, y ejecutarlo con la orden `python programa.py`, el resultado será que se escribirá la cadena de texto 'Hola mundo'.

```
usu@:~$ cat programa.py
print('Hola mundo')
usu@:~$ python programa.py
Hola mundo
usu@:~$
```

Captura de pantalla de terminal de Ubuntu. ([GNU/GPL](#))

Los comentarios en el lenguaje se escriben con el carácter `#`, las asignaciones de valores a las variables con el carácter igual (`=`). Para crear una función utilizamos el siguiente código, hay que tener en cuenta que la sangría del texto sirve para delimitar qué instrucciones van dentro de la función:

```
def nombrefunción(arg1,arg2..):
    instrucción1
    instrucción2
    ...
    instrucciónN
```

Es decir, en este lenguaje no hay delimitador de sentencia como el punto y como de Java o C, ni delimitador de bloques, como las llaves de estos mismos lenguajes.

### Para saber más

Desde el siguiente enlace puedes descargar la versión del intérprete correspondiente a tu sistema operativo:

[Descargar intérprete de Python.](#)

## 3.2.- Declaración de datos. Tipos básicos.

Como ya hemos comentado, en Python no se declaran las variables como tal, sino que se utilizan directamente. Podemos declarar variables de los siguientes tipos básicos:

- ✓ **Números enteros.** Los números enteros son el cero y aquellos números positivos o negativos que no tienen decimales. En Python se pueden representar mediante el tipo `int` que utiliza 32 o 64 bits, dependiendo del tipo de procesador, o el tipo `long`, que permite almacenar números de cualquier precisión, estando limitados solo por la memoria disponible en la máquina.

```
# declaración de un número entero
a = 2
# declaración de un número entero largo
a = 2L
```



Robert Gordon (CC BY-SA)

- ✓ **Números reales.** Son los que tienen la coma decimal. Se expresan mediante el tipo `float`.

```
b = 2.0
```

- ✓ **Números complejos.** Son un tipo de dato especial compuesto por una parte real y una parte imaginaria.

```
complejo = 2.0 + 7.5 j
```

- ✓ **Cadenas.** Es texto encerrado entre comillas simples o dobles.

```
mensaje = "Bienvenido a Python"
```

- ✓ **Booleanos.** Sólo pueden tener dos valores `True` o `False`.

De entre los operadores más importantes para trabajar con estos datos podemos destacar los siguientes:

- ✓ **Operadores aritméticos:** suma (+), resta (-), multiplicación (\*), división(/), división entera (//), exponente (\*\*), módulo (%).
- ✓ **Operadores booleanos:** `and`, `or`, `not`.
- ✓ **Operadores relacionales:** igual (==), distinto (!=), menor (<), mayor (>), menor igual(<=), mayor o igual(>=).

### Autoevaluación

En Python la representación de números enteros depende de la máquina donde se esté ejecutando el programa.

☐ Verdadero ☐ Falso

#### Verdadero

El tipo `int` de Python se implementa a bajo nivel mediante un tipo `long` de C, y dado que Python utiliza C por debajo, como C, el rango de los valores que puede representar depende de la plataforma.

### 3.3.- Estructuras de programación. Colecciones.

Python tiene una serie de estructuras de datos que se utilizan ampliamente. Son tipos avanzados de datos, que reciben el nombre de **Colecciones**, a saber:

- ✔ **Listas.** Son colecciones ordenadas de datos, en otros lenguajes se conocen como arrays o vectores. Pueden tener cualquier tipo de dato.

```
l = [3, True, "mi lista", [ 1 , 2 ] ]
```

Se accede a un elemento concreto de la lista usando el índice de su posición (empezando por cero) entre corchetes.

```
a = l [0]      # a vale 3
b = l [3][1]   # b vale 2
```

- ✔ **Tuplas.** Son parecidas a las listas, con la diferencia que son más ligeras por lo que si el uso es básico se utilizan mejor tuplas. Los elementos van entre paréntesis en lugar de entre corchetes.

```
t = ( "hola", 2, False)
b = t[1]      # b vale 2
```

- ✔ **Diccionarios.** Son colecciones que relacionan una clave y un valor. Para acceder a un valor se utiliza el operador `[]`, igual que para las filas y tuplas, y dentro de él la clave a la que queremos acceder.

```
d = { "Nombre": "Alberto",
      "Apellido": "Contador",
      "Victorias" : [ 2007 , 2008 , 2009 ] }
x = d ["Nombre"]      # x vale "Alberto"
x = d ["Apellido"]    # x vale "Contador"
x = d ["Victorias"][1] # x vale 2008
```

Las listas son **objetos mutables**, es decir, podemos modificar cada uno de sus componentes. Las tuplas sin embargo son **no mutables**, es decir, que no podemos modificarlas una vez que se han creado, por ejemplo, si intentamos hacer esto:

```
tupla = ( 1 , 'hola', 3.0)
tupla [1] = 'adios'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> 008
```

El intérprete de Python nos da un error porque las tuplas no se pueden modificar.



[sunshinecity](#) (CC BY)

## 3.4.- Sentencias del lenguaje.

La sintaxis de las estructuras de programación en Python es la siguiente:

- ✔ **Estructura condicional.** La sintaxis más común de la estructura condicional es un `if` seguido de la condición a evaluar, seguida de dos puntos, y la siguiente línea sangrada, para indicar el código que queremos que se ejecute si se cumple la condición. En caso de que no se cumpla la condición, ejecutamos la sentencia detrás del `else`:

```
if (numero / 2 == 0):  
    print ("El numero es par")  
else:  
    print ("El numero es impar")
```



[anieto2k \(CC BY-SA\)](#)

- ✔ **Bucles.** Nos permiten ejecutar un fragmento de código un número de veces, o mientras se cumpla una determinada condición.

```
while)
```

El bucle `while` (mientras) repite todo su bloque de código mientras la expresión evaluada sea cierta. Detrás de la condición hay que poner dos puntos, y debajo las sentencias que van dentro del `while`.

```
num = 1  
while num <= 10:  
    print (num)  
    num += 1
```

El funcionamiento es sencillo, el programa evalúa la condición, y si es cierta, ejecuta el código que hay dentro del `while`, cuando acaba vuelve a evaluar la condición y si es cierta ejecuta nuevamente el código, y así hasta que la condición devuelva `false`, en cuyo caso, el programa continuaría ejecutando las instrucciones siguientes al bucle. `for ... in`  
La sentencia `for` se utiliza para recorrer secuencias de elementos. Si utilizamos un tipo secuencia, como es la lista en el siguiente ejemplo:

```
milista = ["Maria", "Pepe", "Juan"]  
for elemento in milista:  
    print (elemento )
```

Vemos que el resultado es:

```
Maria  
Pepe  
Juan
```

Lo que hace el bucle es que para cada elemento que haya en la secuencia, ejecuta las línea de código que tenga dentro, en este caso la sentencia `print`. Lo que hace la cabecera del bucle es obtener el siguiente elemento de la secuencia `milista` y almacenarlo en una variable de nombre `elemento`. Por esta razón en la primera iteración del bucle `elemento` valdrá "Maria", en la segunda "Pepe", y en la tercera "Juan".

### Debes conocer

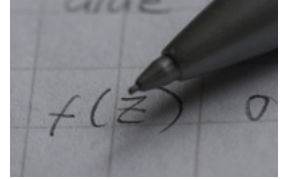
Para mayor información sobre el lenguaje Python puedes consultar el siguiente libro del Departamento de Lenguajes y Sistemas de la Universidad Jaume I:

[Introducción a la programación con Python.](#)

## 3.5.- Llamadas a funciones.

Una función es un conjunto de instrucciones o programa que realiza una tarea específica y que devuelve un valor. Las funciones nos ayudan a reutilizar código, y a dividir el programa en partes más pequeñas con objeto de hacerlo más organizado, legible y más fácil de depurar.

```
def cuadrado ( numero ):  
    print (numero ** 2)
```



[vestimen \(CC BY\)](#)

Con el código anterior estamos declarando la función `cuadrado` que, como su nombre indica, calcula el cuadrado de un número. El número que aparece entre paréntesis es el valor que le pasamos a la función para que pueda calcular su cuadrado, y recibe el nombre de `parámetro`. Las funciones pueden tener uno, varios o ningún parámetro.

Cuando ejecutamos la función se dice que estamos haciendo una llamada a la función, y lo hacemos utilizando su nombre y los parámetros entre paréntesis:

```
cuadrado (4)
```

El resultado de llamar a la función será que se imprimirá por pantalla el valor `16`.

Las funciones pueden o no devolver un valor como resultado de su ejecución. Podemos transformar el ejemplo anterior para que devuelva el resultado en lugar de mostrarlo por pantalla. El código sería el siguiente:

```
def cuadrado ( numero ):  
    return numero ** 2
```

Y llamaríamos a la función con la siguiente instrucción:

```
print (cuadrado(4))
```

El resultado sería el mismo, sólo que ahora la función devuelve el valor, y ese valor es el que muestra por pantalla la instrucción `print`.

### Autoevaluación

**Las funciones en Python, al igual que los procedimientos en otros lenguajes, a veces no devuelven ningún valor.**

☐ Verdadero ☐ Falso

**Falso**

Las funciones siempre devuelven un valor, y cuando el programador no especifica un valor de retorno la función devuelve el valor `None`.



## 3.6.- Clases y objetos.



Shane Global Language Center (CC BY)

Hasta ahora hemos visto la sintaxis básica de Python. Pero para entender bien cómo está formado un módulo en Odoo, necesitamos saber también qué son las **clases** y los **objetos**. Si has programado antes con un lenguaje orientado a objetos, no tendrás ningún problema en entender la definición de ambos conceptos.

Un programa orientado a objetos está formado por clases y objetos. Cuando tenemos un problema que resolver, lo dividimos en una serie de objetos y son esos objetos los que describimos en nuestro programa. El programa consiste en una serie de interacciones entre esos objetos. Las clases son plantillas a partir de las cuales se crean los objetos, cuando creamos un objeto a partir de una clase se dice que ese objeto es una **instancia** de la clase.

Las clases tienen una serie de atributos o campos que las definen, así como una serie de métodos para hacer operaciones sobre esos campos. Cuando creamos un objeto le damos valor a esos atributos. En Python las clases se definen mediante la palabra clave `class` seguida del nombre de la clase, dos puntos (`:`) y a continuación, sangrado, el cuerpo de la clase.

Por ejemplo, vamos a crear una clase `Persona` que tiene como atributos el `nombre` y la `edad`:

```
class persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad
    def mayordeedad(self):
        if self.edad > 18:
            print ("Es mayor de edad" )
        else:
            print ("No es mayor de edad" )
```

En la definición de clase anterior vemos que estamos definiendo un método `__init__`. El nombre de este método no es casual, sino que se utiliza para inicializar el objeto. Cuando instanciamos un objeto a partir de la clase, será el primer método que se ejecutará. También vemos que los dos métodos de la clase tienen como primer argumento `self`. Este argumento sirve para referirnos al objeto actual, y se utiliza para poder acceder a los atributos y métodos del objeto.

Para crear un objeto de la clase utilizamos la siguiente instrucción:

```
empleado = persona("Javier",32)
```

Ahora que ya hemos creado nuestro objeto, podemos acceder a sus atributos y métodos mediante la sintaxis `objeto.atributo` y `objeto.método`:

```
print (empleado.nombre, "tiene", empleado.edad)
```

## 3.7.- Módulos y paquetes.

Cuando salimos del intérprete y volvemos a entrar empezamos desde el principio, si habíamos creado alguna variable o programa éste desaparece y tenemos que volver a escribir el código. Para conservar los programas que hacemos, podemos crear archivos con extensión `.py`, de manera que siempre tengamos el código disponible para ejecutarlo con el intérprete y que no sea necesario volver a introducirlo.

Por otra parte, los programas a veces son muy grandes y al dividirlos en partes se hacen más fáciles de mantener. Se agrupa el código relacionado y se guarda en archivos diferentes. Cada archivo es un módulo en Python. En otras palabras, **un módulo es un archivo que contiene definiciones y declaraciones de Python**.



[Imerio \(CC BY-SA\)](#)

Las razones para utilizar módulos son principalmente las siguientes:

- ✓ Hacer el código más fácil de mantener.
- ✓ Reutilización de código, por ejemplo, una función que queramos utilizar en varios programas.

Si queremos utilizar un módulo en otro archivo, tenemos que importarlo con la siguiente instrucción:

```
import nombre_modulo
```

### Ejercicio resuelto

Crear un módulo `funciones.py` que defina una función que escriba el texto "Bienvenido a Python". Utilizar el módulo en otro que se llame `mi_modulo.py`.

Mostrar retroalimentación

Para crear el módulo lo único que tenemos que hacer es crear un archivo y meter dentro el código de la función, que tal y como hemos visto se define de la siguiente forma:

```
def bienvenida():  
    print("Bienvenido a Python")
```

Hay que tener cuidado con las sangrías, porque en Python es la manera de indicar que una instrucción está dentro de un bloque de código, y puede dar error de ejecución si no están bien definidas.

A continuación tienes en enlace a los archivos de este ejercicio, para ejecutarlo descomprime el archivo en una carpeta y ejecuta el archivo `mi_modulo.py` en el intérprete de Python, con la orden:

```
python mi_modulo.py
```

Al ejecutar el programa, veremos por pantalla el mensaje "Bienvenido a Python"

[Ejercicio resuelto](#) (0.09 MB)

**Un paquete es una colección de módulos relacionados.** Mientras los módulos ayudan a organizar el código, los paquetes ayudan a organizar los módulos. A nivel físico, los módulos son los archivos, y los paquetes son los directorios.

Para hacer que Python trate a un directorio como un paquete es necesario crear un archivo `__init__.py` en dicha carpeta. Para hacer que un módulo pertenezca a un paquete hay que copiar el archivo `nombre_modulo.py` que define el módulo en el directorio del paquete. Para importar paquetes se utiliza la sentencia `import` igual que con los módulos.

### Debes conocer

En el siguiente enlace puedes acceder a más información sobre módulos y paquetes:



## Módulos y paquetes

Elaboración propia (Uso educativo no comercial)

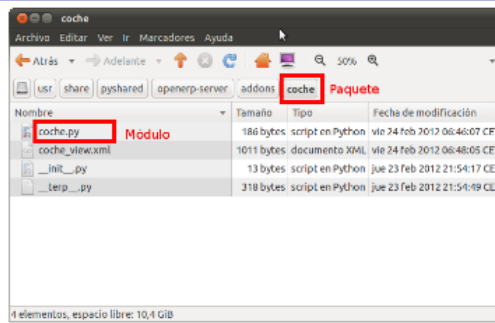
### Definición

**Módulo:** Archivo que contiene definiciones y declaraciones en Python

**Paquete:** Colección de módulos relacionados.

Elaboración propia (Uso educativo no comercial)

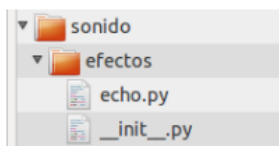
## Ejemplo



- Los módulos son archivos del tipo `nombre_modulo.py`
- Los paquetes son carpetas que incluyen un archivo `__init__.py`

Captura de pantalla de explorador de archivos de Ubuntu. (GNU/GPL)

## Sentencia `import`



- Para utilizar un módulo en otro archivo se utiliza la sentencia `import`
- Podemos importar módulos individuales:  
`import sonido.efectos.echo`
- Subpaquetes:  
`import sonido.efectos`
- O paquetes completos:  
`import sonido`

Captura de pantalla del explorador de archivos de Ubuntu (GNU/GPL)

## 3.8.- Librerías de funciones (APIs).

Una **API**, o **Biblioteca de Clases**, es un conjunto de clases útiles que tienen a disposición los programadores para utilizar en sus programas. Python proporciona una API estándar, cuyo código fuente está a libre disposición para las principales plataformas desde el sitio web de Python. También es posible acceder a muchos módulos libres de terceros en la misma página web, así como a programas, herramientas y documentación adicional.

```
>>> def fib(n):
...     a, b = 0, 1
...     for i in range(n):
...         a, b = b, a+b
...     return a
>>> fib(175)
167244575904137984013222756794978732
>>>
```

Captura de pantalla de terminal de Ubuntu.  
(GNU/GPL)

En el enlace expresado más abajo encontrarás una referencia a toda la Biblioteca estándar de módulos de Python, entre los que destacan:

- ✓ **os**: provee de diferentes funciones del sistema operativo, como creación de archivos y directorios, leer o escribir de un archivo, manipular rutas, etc.
- ✓ **sys**: permite acceder a información sobre el intérprete de Python, como por ejemplo el prompt del sistema, datos de licencia y en general cualquier parámetro o variable que tenga que ver con el intérprete.
- ✓ **datetime**: módulos para la manipulación de fechas y horas.
- ✓ **math**: funciones matemáticas.

### Para saber más

Para mayor información puedes consultar el siguiente enlace donde se describe la Biblioteca estándar que se distribuye con Python:

[Biblioteca de clases estándar de Python.](#)

### Autoevaluación

La función `chdir()` que cambia de directorio la podemos encontrar en el paquete `sys`.

☐ Verdadero ☐ Falso

**Falso**

La función `chdir()` se encuentra en el paquete `os`, de manipulación de funciones del sistema operativo.

## 3.9.- Inserción, modificación y eliminación de datos en objetos.

En este apartado vamos a ver cómo están estructurados los módulos y cómo crear un primer módulo sencillo que cree un objeto en la aplicación para insertar, modificar o eliminar datos en la base de datos.

Los módulos de Odoo se guardan dentro de la carpeta `addons`, que para nuestra versión de Ubuntu se encuentra en `/usr/lib/python3/dist-packages/odoo/addons`. Si has instalado la aplicación desde un archivo de Internet en lugar de desde `Synaptic`, o tu versión de la aplicación o Ubuntu es diferente, puede que ella ruta del directorio `addons` sea diferente.

Lo que tenemos que hacer es crear una carpeta para nuestro módulo dentro del directorio `addons`. Los archivos que tiene que contener la carpeta son:

- ✓ `__init__.py`. Necesario para que la carpeta se trate como un paquete en Python, contiene los import de cada archivo del módulo que contenga código Python, en este caso, el archivo `nombre_modulo.py`.
- ✓ `__manifest__.py`. Contiene un diccionario Python con la descripción del módulo, como quién es el autor, la versión del módulo o cuáles son los otros módulos de los que depende.
- ✓ `nombre_modulo.py`. En este archivo creamos la clase definiendo los campos que va a tener. Al crear la clase estamos creando el modelo (una tabla en la base de datos) y también estamos creando el controlador, porque cuando creamos una clase estamos definiendo el comportamiento que tiene.
- ✓ `nombre_modulo_view.xml`. En este archivo definimos la vista de nuestro módulo, o del objeto que va a crear nuestro módulo. Para crear este archivo necesitamos tener ciertos conocimientos de XML, o bien coger una vista de otro objeto y a partir de ella crear la del nuevo objeto.

También existe otra forma de crear módulos sin necesidad de ningún desarrollo, que es utilizando el módulo `base_module_record`. Si instalamos este módulo, tendremos la opción de grabar todas las acciones que realicemos en la aplicación. Si has utilizado alguna vez las macros en un procesador de textos o en una hoja de cálculo sabrás de lo que estamos hablando. El módulo `base_module_record` funciona de forma parecida. Por ejemplo, podemos crear un nuevo objeto y asignarle un menú, todo ello se grabaría en un archivo comprimido. Realmente ese archivo comprimido lo que va a contener es un módulo, con todos los archivos indicados anteriormente. Si con posterioridad cogemos esos archivos y los copiamos a la carpeta `addons`, podremos instalarlos en la aplicación como cualquier otro módulo. Al instalarlo, el resultado sería que aparecería ese nuevo objeto y menú en nuestra aplicación.

### Debes conocer

En el siguiente enlace de la página web de Odoo puedes consultar como se construye un módulo en Odoo:

[Crear un módulo en Odoo](#)

## 4.- Entornos de desarrollo y herramientas de desarrollo en sistemas ERP-CRM.

### Caso práctico



[pexels \(CC0\)](#)

**Juan** está buscando un entorno de desarrollo para la creación de los módulos. Ya ha empezado a programar, el código de los módulos es bastante extenso y le resulta tedioso tener que trabajar con el intérprete. Necesita un entorno de desarrollo que le ayude con la sintaxis del lenguaje, y a depurar los programas.

La herramienta más importante cuando estamos programando siempre es un buen entorno de desarrollo. Si queremos una respuesta rápida y el programa es pequeño podemos utilizar el intérprete, como hemos venido haciendo hasta ahora. Sin embargo, cuando el programa va siendo más grande, es mucho más cómodo utilizar un entorno de desarrollo desde el cual podemos hacer todas las operaciones de manera gráfica.



[www.python.org](http://www.python.org) (GNU/GPL)

Un entorno de desarrollo es un programa que incluye todo lo necesario para programar en uno o varios lenguajes. Además de un editor de textos especializado como herramienta central, incluyen navegadores de archivos, asistentes de compilación, depuradores, etc.

Entre los entornos de desarrollo para trabajar con Python nos encontramos con uno de los más sencillos, **IDLE**, disponible en la mayoría de las plataformas. En Ubuntu se puede descargar desde Synaptic. Este entorno básicamente lo que permite es revisar la sintaxis de nuestro módulo y ejecutarlo. También incluye opciones de depuración.

Cuando lo ejecutamos nos aparece una ventana con un sencillo menú. Desde él podremos abrir el archivo con código python, y aparecerá una nueva ventana donde podremos revisar la sintaxis desde el menú Run/Check module o ejecutar el programa desde el menú Run/Run module, entre otras opciones.

En otro orden de cosas, tenemos la **extensión o plugin de Odoo para el editor Gedit**. Esta extensión permite utilizar el editor de texto como un entorno de desarrollo para Odoo. La principal ventaja es que añade fragmentos de código típico, lo cual facilita en gran medida no tener que recordar la sintaxis.

Finalmente, dentro de los entornos de desarrollo más completos para trabajar con Python tenemos **Eclipse**. Eclipse además incorpora una serie de plantillas con código, lo cual facilita mucho la creación de los módulos en Odoo.

### Para saber más

En este enlace se explica cómo instalar el entorno de desarrollo **Eclipse** y Pydev para trabajar con Odoo:

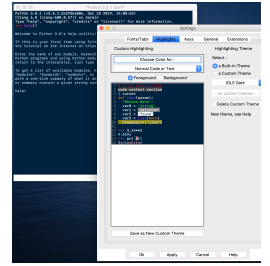
[Instalar Eclipse para trabajar con Odoo.](#)

## 4.1.- Depuración de un programa.

Las herramientas de depuración cuando estamos desarrollando un programa ayudan a detectar errores en el código, mostrando información de qué tipo de error es o porqué se produce. Las herramientas de depuración suelen formar parte de los entornos de desarrollo.

El lenguaje Python incorpora un depurador dentro de su Biblioteca de módulos estándar llamado `pdb`. Soporta puntos de ruptura y ejecución paso a paso del código, inspeccionar valores de variables y otras opciones de depuración. Sin embargo, como en todos los lenguajes, si utilizamos una herramienta gráfica para el proceso de depuración, seguramente ahorraremos mucho tiempo.

Dentro del entorno de desarrollo `IDLE` podemos depurar nuestros programas. Este entorno está disponible para cualquier distribución y sistema operativo. Para activar el módulo de depuración hacemos clic en el menú `Debug/Debugger`. Una vez que esté el modo de depuración activado el funcionamiento es como el de cualquier depurador. Con el botón derecho establecemos una parada en la sentencia del código que queremos evaluar, de manera que al ejecutar el programa el depurador se parará en esa instrucción, y podremos analizar el valor de las variables en ese punto.



[Mačko \(CC BY-SA\)](#)

Las acciones que podemos realizar con el depurador son las siguientes:

- ✓ **Go:** hace que la ejecución del programa continúe hasta que encuentre un punto de ruptura.
- ✓ **Step:** va ejecutando el programa línea a línea.
- ✓ **Over:** hace que la sentencia actual sea completamente ejecutada sin parar en ninguna función anidada.
- ✓ **Out:** hace que se compute desde el punto actual hasta el fin de la función actual y que esta se termine.
- ✓ **Quit:** finaliza la ejecución del programa.

### Para saber más

En la documentación de la Biblioteca estándar de Python podemos encontrar información sobre el entorno de desarrollo `IDLE`:

[Información sobre IDLE en la Biblioteca estándar de Python.](#)

### Autoevaluación

**Los programas de depuración por definición siempre se distribuyen junto con los entornos de desarrollo.**

☐ Verdadero ☐ Falso

**Falso**

Aunque lo más común es encontrarlos dentro de los entornos de desarrollo, existen programas que son únicamente depuradores, como por ejemplo `pdb` de Python.



## 4.2.- Manejo de errores.



Cuando nuestro programa falla, se genera un error en forma de excepción. Las excepciones son errores detectados por el intérprete durante la ejecución del programa. Por ejemplo, podemos intentar dividir por cero o acceder a un archivo que no existe, entonces el programa genera una excepción avisando al usuario de que se ha producido un error. Nuestra labor como programadores es escribir el código de manera que se contemplen esas posibles excepciones, y se actúe en consecuencia. Si nuestro programa no contempla el manejo de excepciones, cuando ocurra alguna lo que pasará es que se generará el error y el programa dejará de ejecutarse o no tendrá el funcionamiento esperado.

En Python se utiliza una construcción `try-except` para capturar y tratar las excepciones. Dentro del bloque `try` se sitúa el código que creemos que podría producir una excepción, y dentro del bloque `except`, escribimos el código que se ejecutará si se produce dicha excepción.

Veamos un pequeño programa que lanzaría una excepción al intentar hacer una división entre 0:

```
def dividir(a, b):  
    return a / b  
dividir(3,0)
```

Si lo ejecutamos en el intérprete obtenemos la siguiente salida de error:

```
ZeroDivisionError: integer division or modulo by zero
```

Lo que nos dice el mensaje es que es una excepción, `ZeroDivisionError`, y la descripción del error: "integer division or modulo by zero" (módulo o división entera entre cero).

Si escribimos el mismo programa, pero teniendo en cuenta el manejo de excepciones, el código quedaría como sigue:

```
try:  
    def dividir(a, b):  
        return a / b  
    dividir(1,0)  
except:  
    print ("Ha ocurrido un error")
```

### Para saber más

Si quieres aprender más sobre excepciones puedes consultar el siguiente tutorial de Python:

[Información sobre excepciones en Python.](#)

## 5.- Formularios e informes en sistemas ERP-CRM.

### Caso práctico



Elaboración propia (Uso educativo no comercial)

**María** está repasando conceptos de XML para generar las vistas del modelo. Concretamente está analizando cómo están formadas las vistas de los módulos que vienen instalados con la aplicación. Lo primero que ha hecho es hacerse una plantilla XML, donde ha incluido todo el código necesario para crear un menú, una acción y dos vistas formulario y árbol para el objeto que incluye el módulo. A partir de ahora utilizará esa plantilla, y sólo tendrá que cambiar los nombres de los elementos para generar vistas nuevas de un objeto.

Los **formularios** constituyen la interfaz de un módulo. Una vez creado el objeto del módulo, o sea, el archivo `nombre_modulo.py`, lo que debemos hacer es crear los menús, acciones, vistas (formulario u otro tipo) para poder interactuar con el objeto. Estos elementos se describen en el archivo `nombre_modulo_view.xml`.

Decir que es importante recordar que el nombre del fichero XML se encuentra listado en el `__manifest__.py`.

Los **informes** pueden ser estadísticos, para los que tenemos el módulo `base_report_creator` que permite crear listados, informes y gráficos por pantalla, y pueden ser informes impresos, en cuyo caso existen diversas herramientas para generarlos.

```
odoo@serverodoo:~/modules/base_aabbs$ ls -la
total 44
drwxr-xr-x 8 odoo odoo 4096 ago 7 14:12 .
drwxr-xr-x 5 odoo odoo 4096 ago 7 09:41 ..
-rw-r--r-- 1 odoo odoo 689 ago 7 11:56 1
drwxr-xr-x 3 odoo odoo 4096 ago 7 12:54 controllers
drwxr-xr-x 2 odoo odoo 4096 ago 7 09:41 demo
-rw-r--r-- 1 odoo odoo 71 ago 7 09:41 __init__.py
-rw-r--r-- 1 odoo odoo 944 ago 7 11:52 __manifest__.py
drwxr-xr-x 3 odoo odoo 4096 ago 7 12:56 models
drwxr-xr-x 2 odoo odoo 4096 ago 7 14:10 reports
drwxr-xr-x 2 odoo odoo 4096 ago 7 09:41 security
drwxr-xr-x 2 odoo odoo 4096 ago 7 13:24 views
```

Captura de pantalla de terminal de Ubuntu.  
([GNU/GPL](#))

## 5.1.- Arquitectura de formularios e informes. Elementos.

Los formularios y demás vistas de la aplicación son construidos de forma dinámica por la descripción XML de la pantalla del cliente. Podemos decir que el lenguaje XML es un metalenguaje que utiliza etiquetas del tipo `etiqueta_contenido_fin etiqueta` para expresar información estructurada.

Una etiqueta consiste en una marca hecha en el documento, que hace referencia a un elemento o pedazo de información. Las etiquetas tienen la forma `<nombre atributos>contenido</nombre>`, donde `nombre` es el nombre del elemento que se está señalando.



Captura de pantalla de terminal de Ubuntu.  
(GNU/GPL)

En apartados anteriores has tenido acceso a un breve manual sobre el metalenguaje XML. Si lo necesitas puedes echarle un vistazo para comprender mejor los conceptos de este apartado.

En el caso de Odoo, la estructura del archivo `nombre_modulo_view.xml` y el fichero de definición `__manifest__.py` es la siguiente:

*openacademy/\_\_manifest\_\_.py*

```
'data': [
    # 'security/ir.model.access.csv',
    'templates.xml',
    'views/openacademy.xml',
],
# only loaded in demonstration mode
'demo': [
```

*openacademy/views/openacademy.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<odoo>

<!-- window action -->
<!--
    The following tag is an action definition for a "window action",
    that is an action opening a view or a set of views
-->
<record model="ir.actions.act_window" id="course_list_action">
    <field name="name">Courses</field>
    <field name="res_model">openacademy.course</field>
    <field name="view_mode">tree,form</field>
    <field name="help" type="html">
        <p class="o_view_nocontent_smiling_face">Create the first course</p>
    </field>
</record>

<!-- top level menu: no parent -->
<menuitem id="main_openacademy_menu" name="Open Academy"/>
<!-- A first level in the left side menu is needed
    before using action= attribute -->
<menuitem id="openacademy_menu" name="Open Academy"
    parent="main_openacademy_menu"/>
<!-- the following menuitem should appear *after*
    its parent openacademy_menu and *after* its
    action course_list_action -->
<menuitem id="courses_menu" name="Courses" parent="openacademy_menu"
    action="course_list_action"/>

<!-- Full id location:
    action="openacademy.course_list_action"
```

It is not required when it is the same module -->

```
</odoo>
```

Cada tipo de registro, hace referencia a un objeto diferente. Los registros se definen con la etiqueta `<record>` `</record>`, que indican el inicio y fin de la descripción del registro. Dentro van los campos `field` que se utilizan para definir las características del registro. Siguiendo la estructura anterior, un ejemplo de creación de **vista de tipo formulario** sería el siguiente:

```
<record model="ir.ui.view" id="view_agenda_form">
  <field name="name">agenda</field>
  <field name="model">agenda</field>
  <field name="type">form</field>
  <field name="priority" eval="5"/>
  <field name="arch" type="xml">
    <form string="Agenda">
      <field name="nombre" select="1"/>
      <field name="telefono" select="1" />
    </form>
  </field>
</record>
```

El campo `name` hace referencia al nombre de la vista y el campo `name="model"` hace referencia al objeto del modelo, es decir, la tabla en la base de datos, del cual va a mostrar la información. Vemos que la vista está formada por dos campos: `nombre` y `telefono`.

De igual forma podemos definir el **elemento de menú** para acceder al objeto agenda:

```
<menuitem name="Agenda" id="menu_agenda_agenda_form"/>
<menuitem name="Agenda" id="menu_agenda_form" parent="agenda.menu_agenda_agenda_form" action="action_agenda_form"/>
```

Así como la **acción asociada a ese elemento de menú**:

```
<record model="ir.actions.act_window" id="action_agenda_form">
  <field name="res_model">agenda</field>
  <field name="domain">[]</field>
</record>
```

## Autoevaluación

En la definición de una vista `<record>`, con la etiqueta podemos crear acciones y vistas.

☐ Verdadero ☐ Falso

**Verdadero**

Con el atributo `model` definimos si se trata de una acción o de una vista.

## 5.2.- Herramientas para la creación de formularios e informes.

---



[pxfuel](#) (CC0)

Existen diferentes herramientas para la creación de formularios e informes en Odoo. Para empezar hay que distinguir entre la creación de formularios y la creación de informes. La creación de formularios como ya hemos visto se realiza en archivos `.....XML` con el nombre `nombre_modulo_view.xml`. La creación de informes se realiza con otras herramientas. Ya hemos visto cómo crear informes con el módulo `base_report_creator` y con la extensión de OpenOffice.org para Odoo. Existe otra forma de crear informes, y es mediante la utilización de la librería Jasper Reports.

Jasper Reports es una librería para la generación de informes. Su funcionamiento consiste en generar un archivo XML con la descripción del informe a crear, que es tratado para generar un documento en un formato de salida, como por ejemplo `PDF` o `HTML`.