



Formación técnica – Soluciones

Versión 7.0

Contenido

1. Acerca de OpenERP	
1.1. Punto de vista funcional	
1.2. Punto de vista técnico	
1.3. Instalación en una máquina Unix con fines de desarrollo	
Bazar	
Descarga e instalación de OpenERP	
2. Introducción a Python	
2.1. Interpretador Python	
2.2. Números	
2.3. Cadenas	
2.4. If	
2.5. For	
2.6. Listas	
Compresión de listas	
2.7. Definición de funciones	
Parámetros por defecto	
Lambda Formularios	
Lista de argumentos arbitrarios	
2.8. Las tuplas	
2.9. Conjuntos	
2.10. Diccionarios	
2.11. Módulos	
2.12. Clases	
Sintaxis de definición de clases	
Objetos de clase	
Herencia	
3. Configuración	
3.1. Open Source RAD con OpenObject	
3.2. Instalación de OpenERP	
OpenERP Arquitectura	
3.3. Instalación de paquetes	
3.4. Instalación desde el código fuente	
Procedimiento típico bazar checkout	
3.5. Creación de base de datos	
4. Construir un modulo OpenERP	
4.1. Composición de un modulo	
4.2. Estructura del modulo	
4.3. Servicio Object-ORM	
4.4. Tipos de campo ORM	
Atributos compatibles con los campos	
4.5. Nombres de los campos especiales/reservados	
4.6. Acciones y menús	
5. Construcción de vistas	
5.1. Declaración de vista genérica	
5.2. Vista Tree (Árbol)	
5.3. Vista Form (Formulario)	

6. Relaciones entre Objetos	
6.1. Campos relacionales.	
7. Herencia	
7.1. Mecanismos de herencia	
7.2. Herencia de vista	
Domains (Dominios)	
8. Métodos	
8.1. Funciones en campos	
8.2. Onchange	
9. Vistas Avanzadas	
9.1. Lista y Tree	
9.2. Calendario	
9.3. Buscar vistas	
9.4. Gráficos	
9.5. Tablero Kanban	
10. Flujos de Trabajo	
11. Seguridad	
11.1. Grupos basados en mecanismos de control de acceso	
11.2. Derecho de acceso	
11.3. Nomas de registro	
12. Wizards	
12.1. Objetos Wizards	
12.2. Ejecución de Wizard	
13. Internacionalización	
14. Reportes	
14.1. Impresión de reportes	
Expresiones utilizadas en las plantillas de reportes OpenERP	
Reportes RML	
Reportes WebKit	
14.2. DashBoards	
15. WebServices	
15.1. Biblioteca XML-RPC	
15.2. Biblioteca OpenERP Client	

Basado en un ejemplo real, este manual contiene: la elaboración de un módulo de OpenERP con sus respectivas interfaz, Vistas, Los informes, flujos de trabajo, los aspectos de seguridad, Wizards, WebServices, internacionalización, Rapid Application Development (RAD) y optimización del rendimiento.

1. Acerca de OpenERP

1.1. Punto de vista funcional

- Mostrar OpenERP
- Mostrar principales conceptos funcionales en una db demostración
 - Menús
 - Vistas
 - ✓ Lista + Buscar
 - ✓ Forma
 - ✓ Calendario, Gráfico, etc.
 - Módulos
 - Usuarios
 - ¿Empresas?
 - Objetos principales (ide una manera funcional!)
 - ✓ Partner (socio)
 - ✓ Producto

1.2. Punto de vista técnico

- **Estructura general, el esquema**
 - Servidor - Cliente - Cliente Web
 - Comunicación: Net-RPC, XML-RPC
- **Framework – ORM**
 - ¿Qué es un ORM? - Objeto, relaciones entre objetos
 - OpenERP ORM particularidades
 - ✓ 1 instancia de un objeto de la lógica no es el 1 de registro en la base de datos
 - ✓ 1 instancia de un objeto de la lógica es 1 mesa en el DB
 - ✓ Breve resumen del objeto OSV, métodos ORM

1.3. Instalación en una máquina Unix con fines de desarrollo

Bazaar

- Versiones: principio general
- Sistema de control de versiones centralizado vs sistema de control de versiones distribuido
- Comandos BZR principales

Descarga e instalación de OpenERP

La instalación de la versión OERP en Linux:

- **Introducción a bzt primero**
 - Instalación Bzt
 - Comandos principales
 - Detalles sobre cómo van a utilizarse
- **Descarga de Server, Client, Complementos**
 - Observaciones sobre extra-addons, comunidad-addons

- **Instalación de Dependencias**

- PostgreSQL: Instalación, añadir nuevos usuarios
- OpenERP configuración: cómo crear OpenERP-serverrc
- Python dependencias

2. Introducción ha Python

Python es un lenguaje de indotación sensible imperativo / orientado a objetos, utilizando características de declaración/lenguajes funcionales (estas características están fuera del alcance de la formación actual). Para poder acceder al tutorial de python oficial ingresa a: <http://docs.python.org/tutorial/index.html>

2.1. Interpretador Python

Al escribir un carácter al final de un archivo (Control-D en Unix, Control-Z en Windows) en el indicador principal causa que el intérprete de un código de salida de cero. Si eso no funciona, puede salir del intérprete tecleando el siguiente comando: *quit ()*.

En el modo interactivo que solicita el comando siguiente con el mensaje principal, usualmente tres signos mayor que (>>>), Para las líneas de continuación se le pedirá con el indicador secundario, por defecto tres puntos (...).

Se necesitan líneas de continuación al entrar en una construcción de varias líneas. A modo de ejemplo, echar un vistazo a esta sentencia if:

```
>>> el_mundo_es_plano = 1
>>> if el_mundo_es_plano:
...     print "Ten cuidado de no caerte!"
...
...
Ten cuidado de no caerte!
```

(Ejemplo de un if)

En los sistemas Unix tipo BSD, los scripts de Python se pueden hacer directamente ejecutable, como los scripts de shell, poniendo la línea.

```
#!/usr/bin/env python
```

(Suponiendo que el intérprete está en el PATH del usuario) al comienzo de la secuencia de comandos y dar al archivo un modo ejecutable.

2.2. Números

El intérprete actúa como una simple calculadora: se puede escribir una expresión en ella y se escribe el valor. La sintaxis es sencilla: los operadores +, -, * y / funcionan igual que en la mayoría de otros idiomas (por ejemplo, Pascal o C); paréntesis se pueden utilizar para la agrupación. Por ejemplo:

```

>>> 2+2
4
>>> # Esto es un comentario
... 2+2
4
>>> 2+2 # Y esto es un comentario en la misma línea de un código
4
>>> (50-5*6)/4
5
>>> # División de enteros
... 7/3
2
>>> 7/-3
-3

```

El signo igual ('=') se utiliza para asignar un valor a una variable. Posteriormente, ningún resultado se muestra antes del siguiente modo interactivo:

```

>>> width = 20
>>> height = 5*9
>>> width * height
900

```

Un valor se puede asignar a varias variables simultáneamente:

```

>>> x = y = z = 0 # X, Y, Z valen Zero
>>> x
0
>>> y
0
>>> z
0

```

Las variables deben ser "definidas" (asignarles un valor) antes de que puedan ser utilizadas, o se producirá un error:

```

>>> # Acceder a una variable no definida
... n
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'n' is not defined

```

Hay compatibilidad total para el punto flotante (float); realizar operaciones mixtas, convirtiendo una operación entera (integer) a una flotante (float):

```

>>> 3 * 3.75 / 1.5
7.5
>>> 7.0 / 2
3.5

```

También pueden realizarse funciones de conversión de flotantes a enteros (float (), int () y long ()).

2.3. Cadenas

Además de números, Python también puede manipular cadenas, que se puede expresar de varias maneras. Ellas pueden ir entre comillas simples o dobles:

```
>>> 'spam eggs' #Comillas Simples
'spam eggs'
>>> 'doesn\'t'
"doesn't"
>>> "doesn't" #Comillas Dobles
"doesn't"
>>> '"Yes," he said.' #Comillas Simples y Dobles
```

Las cadenas pueden ser concatenadas (pegadas) con el operador +, y se repiten con el operador *:

```
>>> Mundo = 'Ayuda' + 'A'
>>> Mundo
'AyudaA'
>>> '<' + word*5 + '>'
'<AyudaAAyudaAAyudaAAyudaAAyudaA>'
```

Dos cadenas juntas una a la otra se concatenan automáticamente, la primera línea de arriba también podría haber sido escrita palabra = 'Ayuda' 'A', esto sólo funciona con dos literales, no con expresiones de cadena arbitrarias:

```
>>> 'computa' 'dora' # <- Expresion valida
'computadora'
>>> 'computa'.strip() + 'dora' # <- Expresion valida
'computadora'
>>> 'computa'.strip() 'dora' # <- Expresion no valida
```

Las cadenas pueden ser subíndices (indexado); como en C, el primer carácter de una cadena tiene el (índice) 0. No hay ningún tipo de carácter independiente, un personaje es simplemente una cadena de longitud uno. Al igual que en el Icono, sub-cadenas se pueden especificar con la notación de corte: dos índices separados por dos puntos.

```
>>> Mundo[4]
'a'
>>> Mundo[0:2]
'Au'
>>> Mundo[2:4]
'ua'
```

Los índices pueden ser números negativos, para empezar a contar desde la derecha. Por ejemplo:

```
>>> Mundo[-1] # El ultimo caracter de la cadena
'A'
>>> Mundo[-2] # El Penultimo caracter de la cadena
'a'
>>> Mundo[-2:] # Los dos últimos caracteres
'aA'
>>> Mundo[:-2] # Todos, excepto los dos últimos caracteres de la cadena
'Ayud'
```

NOTA: Pero tenga en cuenta que -0 es lo mismo que 0, por lo que no se cuenta desde la derecha.

```
>>> Mundo[-0] # (desde -0 es igual a 0)
'A'
```

Cadenas y objetos de código único (unicode) tienen una única función de operación: el operador `%` (módulo). Esto también es conocido como el formato de cadenas u operador de interpolación. Teniendo en cuenta los valores de `%` de formato (donde formato es una cadena o un objeto Unicode), especificaciones de conversión `%` en formato se sustituyen por cero o más elementos de los valores.

Si el formato requiere un solo argumento, los valores pueden ser un único objeto no tupla. De lo contrario, los valores deben ser una tupla con exactitud el número de elementos especificados por la cadena de formato, o un solo objeto de asignación (por ejemplo, un diccionario). Los posibles valores son:

`'%d'` Firmado decimal entero.

`'%e'` Flotando formato exponencial punto (en minúsculas).

`'%E'` Flotando formato exponencial punto (en mayúsculas).

`'%f'` Formato decimal de coma flotante.

`'%F'` Formato decimal de coma flotante.

`'%c'` Carácter individual (acepta cadena de caracteres entero o individual).

`'%s'` String (convierte cualquier objeto de Python usando `str()`).

He aquí un ejemplo.

```
>>> import math
>>> print 'El valor de PI es aproximadamente %5.3f.' % math.pi
El valor de PI es aproximadamente 3.142.
```

2.4.If

Tal vez el tipo más conocido de declaración es la instrucción `if`. Por ejemplo:

```
>>> x = int(raw_input("Por favor introduzca un número entero: "))
Por favor introduzca un número entero: 42
>>> if x < 0:
...     x = 0
...     print 'Negativo cambiado a cero'
... elif x == 0:
...     print 'Zero'
... elif x == 1:
...     print 'Simple'
... else:
...     print 'Mas'
...
Mas
```

Puede haber cero o más partes `elif` y la parte `else` es opcional. La palabra clave `'elif'` es la abreviatura de `'else if'`.

2.5. For

La declaración en Python difiere un poco de lo que puede estar acostumbrado en C o Pascal. En lugar de siempre iterar sobre una progresión aritmética de números (como en Pascal), o dando al usuario la capacidad de definir tanto la etapa de iteración y condiciones de detención (como C), la instrucción `for` en Python itera sobre los elementos de cualquier secuencia (una lista o una cadena), en el orden en que aparecen en la secuencia. Por ejemplo (sin juego de palabras):

```
>>> # Medir algunas cadenas:
... a = ['gato', 'ventana', 'carro']
>>> for x in a:
...     print x, len(x)
...
gato 4
ventana 7
carro 5
```

No es seguro modificar la secuencia que se repiten a lo largo del ciclo (esto sólo puede suceder por tipos secuenciales mutables, como las listas).

2.6. Listas

Python conoce una serie de tipos de datos compuestos, que se utilizan para agrupar otros valores. El más versátil es la lista, que se puede escribir como una lista de valores separados por comas (material) entre corchetes. Nombrar el material no es necesario que todos tienen el mismo tipo.

```
>>> a = ['carro', 'casa', 100, 1234]
>>> a
['carro', 'casa', 100, 1234]
```

Al igual que los índices de cadenas, listas índices comienzan en 0, y las listas se pueden cortar, concatenar y así sucesivamente.

Diferencia de las cadenas, que son inmutables, es posible cambiar los elementos individuales de una lista.

```
>>> a
['carro', 'casa', 100, 1234]
>>> a[2] = a[2] + 23
['carro', 'casa', 123, 1234]
```

Asignación por partes también es posible, y esto puede incluso cambiar el tamaño de la lista o borrar por completo:

```

>>> # Reemplace algunos items
... a[0:2] = [1, 12]
>>> a
[1, 12, 123, 1234]
>>> # Quiramos algunos:
... a[0:2] = []
>>> a
[123, 1234]
>>> # Insertamos algunos:
... a[1:1] = ['carro', 'casa']
>>> a
[123, 'carro', 'casa', 1234]
>>> #Insertar (una copia) al principio
>>> a[:0] = a
>>> a
[123, 'carro', 'casa', 1234, 123, 'carro', 'casa', 1234]
>>> # Borrar la lista: reemplazar todos los elementos con una lista vacía
>>> a[:] = []
>>> a
[]

```

La función `len()` también se aplica a las listas:

```

>>> a = ['a', 'b', 'c', 'd']
>>> len(a)
4

```

list.append (x) Añade un elemento al final de la lista, equivalente al `[len (a):] = [x]`.

list.Count (x) Devuelve el número de veces que `x` aparece en la lista.

list.Sort () Ordene los elementos de la lista, en su lugar.

list.reverse () Invertir la lista de elementos, que se colocan desde la última a la primera.

Compresión de listas

Las comprensión de listas proporcionan una manera concisa de crear listas sin tener que recurrir al uso de `map()`, `filter()` y / o `lambda`. La definición de lista resultante tiende a menudo a ser más claro que las listas construidas con esas construcciones. Cada lista por comprensión consiste en una expresión seguida de una cláusula, a continuación, cero o más a favor o en caso de cláusulas.

El resultado será una lista, como resultado de la evaluación de la expresión en el contexto del *for* e *if* que le siguen. Si la expresión sería evaluar a una tupla, se debe colocar entre paréntesis.

```

>>> freshfruit = [' banana', ' frambuesa ', ' granadina ']
>>> [weapon.strip() for weapon in freshfruit]
['banana', 'frambuesa', 'granadina']
>>> vec = [2, 4, 6]
>>> [3*x for x in vec]
[6, 12, 18]
>>> [3*x for x in vec if x > 3] #Si los valores de x son mayores a 3 multiplique el valor
[12, 18]
>>> [3*x for x in vec if x < 2] #Si los valores de x son mayores a 3 multiplique el valor
[]
>>> [[x,x**2] for x in vec] #Utilizacion de corchete asociacion de funciones
[[2, 4], [4, 16], [6, 36]]

>>> [x, x**2 for x in vec] # error - paréntesis necesarios para tuplas
File "<stdin>", line 1, in ?
[x, x**2 for x in vec]
    ^
SyntaxError: invalid syntax

```

(Ejemplo 1 compresión de listas)

```

>>> [(x, x**2) for x in vec]
[(2, 4), (4, 16), (6, 36)]
>>> vec1 = [2, 4, 6]
>>> vec2 = [4, 3, -9]
>>> [x*y for x in vec1 for y in vec2] #Multiplicamos los valores de vec1 con cada uno de los valores de vec2
[8, 6, -18, 16, 12, -36, 24, 18, -54]
>>> [x+y for x in vec1 for y in vec2] #Sumamos los valores de vec1 con cada uno de los valores de vec2
[6, 5, -7, 8, 7, -5, 10, 9, -3]
>>> [vec1[i]*vec2[i] for i in range(len(vec1))] # Multiplicamos los atributos del vec1 con el vec2 en base a su posición
[8, 12, -54]

```

(Ejemplo 2 compresiones de listas)

2.7.Compresión de listas

Podemos crear una función que escriba la serie de Fibonacci hasta un límite arbitrario. Es fácil de escribir una función que devuelve una lista de los números de la serie de Fibonacci, en lugar de imprimirlo.

```

>>> def fib2(n): # retorno serie de Fibonacci hasta n
...     """Devuelve una lista que contiene la serie de Fibonacci hasta n"""
...     result = []
...     a, b = 0, 1
...     while a < n:
...         result.append(a)
...         a, b = b, a+b
...     return result
...
>>> f100 = fib2(100) #Llamamos
>>> f100 # Escribir el resultado
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]

```

Parámetros por defecto

La forma más útil es especificar un valor predeterminado para uno o más argumentos. Esto crea una función que puede ser llamada con menos argumentos de lo que se define para permitir. Por ejemplo:

```
def ask_ok(prompt, retries=4, complaint='Yes or no, please!'):
    while True:
        ok = raw_input(prompt)
        if ok in ('y', 'ye', 'yes'):
            return True
        if ok in ('n', 'no', 'nop', 'nope'):
            return False
        retries = retries - 1
        if retries < 0:
            raise IOError('refusenik user')
        print complaint
```

Las funciones también pueden ser llamados con argumentos clave de la forma clave = valor.

```
ask_ok(retries=5 ask="What do we do?")
```

Lambda

Por demanda popular, una serie de características que se encuentran comúnmente en los lenguajes de programación funcionales como Lisp se han añadido a Python. Con la palabra clave lambda, se pueden crear pequeñas funciones anónimas. He aquí una función que devuelve la suma de sus dos argumentos: lambda a, b: a + b.

Las formas lambda pueden utilizarse siempre que se requieren los objetos de función. Están sintácticamente restringidas a una sola expresión. Semánticamente, no son más que otra manera de expresar una definición de función normal. Al igual que la definición de funciones anidadas, las formas lambda pueden hacer referencia a variables del ámbito contenedor.

```
>>> def make_incrementor(n):
...     return lambda x: x + n
...
>>> f = make_incrementor(42)
>>> f(0)
42
>>> f(1)
43
```

Listas arbitrarias de argumentos

Por último, la opción que se utiliza con menor frecuencia es especificar que una función puede ser llamada con un número arbitrario de argumentos. Estos argumentos serán envueltos en una tupla (ver Tuplas y secuencias). Antes del número variable de argumentos, pueden aparecer cero o más argumentos normales.

```
def write_multiple_items(file, separator, *args):
    file.write(separator.join(args))
```

2.8. Tuplas

Las tuplas, como las cadenas, son inmutables: no es posible asignar a los elementos individuales de una tupla (se puede simular la mayor parte del mismo efecto con corte

y concatenación, sin embargo). También es posible crear tuplas que contienen objetos mutables, como las listas.

2.9. Sets

Python also includes a data type for sets. A set is an unordered collection with no duplicate elements.

```
>>> a = set('abracadabra')
>>> a #Mostrar por letra
set(['a', 'r', 'b', 'c', 'd'])
```

2.10. Diccionarios

Otro tipo de datos útiles incorporado en Python son los diccionarios. Los diccionarios se encuentran a veces en otros idiomas como "memorias asociativas" o "matrices asociativas". A diferencia de las secuencias, que están indexadas en un rango de números, los diccionarios se indexan por teclas, que puede ser cualquier tipo inmutable; cadenas y números siempre pueden ser teclas. Las tuplas pueden usarse como claves si sólo contienen cadenas, números o tuplas, si una tupla contiene cualquier objeto mutable directo o indirectamente, no puede ser utilizado como una clave.

Lo mejor es pensar en un diccionario como un conjunto desordenado de parejas clave: valor, con el requisito de que las claves son únicas (dentro de un diccionario).

Un par de llaves crea un diccionario vacío: {}. La colocación de una lista separada por comas de parejas *clave: valor* dentro de las llaves añade clave inicial: pares de valores al diccionario, lo que es también la forma en diccionarios se escriben en la salida.

He aquí un pequeño ejemplo de cómo usar un diccionario:

```
>>> tel = {'jack': 4098, 'sape': 4139} #Creamos el diccionario
>>> tel['guido'] = 4127             #Agregamos conjunto
>>> tel
{'sape': 4139, 'guido': 4127, 'jack': 4098} #El diccionario mas el nuevo conjunto
>>> tel['jack']
4098
>>> del tel['sape']                 #Eliminamos el conjunto
>>> tel['irv'] = 4127               #Agregamos conjunto
>>> tel
{'guido': 4127, 'irv': 4127, 'jack': 4098} #El diccionario mas el nuevo conjunto y sin el campo eliminado
>>> tel.keys()                     #Que muestre solo las clave de los conjuntos
['guido', 'irv', 'jack']
>>> 'guido' in tel                 #Que diga si el conjunto guido pertenece al diccionario tel
True
```

Cuando un bucle recorre a través de los diccionarios, el valor de clave correspondiente se puede recuperar en el mismo método de tiempo utilizando los iteritems ().

```
>>> caballeros = {'jose': 'el inteligente', 'jesus': 'el audaz'} #Creas el diccionario
>>> for k, v in caballeros.iteritems():
...     print k, v                #Indicas que imprima cada valor por separado
...
jose el inteligente
jesus el audaz
```

Cuando usas un bucle a través de una secuencia, el índice de posición y el valor correspondiente se pueden recuperar al mismo tiempo utilizando la función `enumerate()`.

```
>>> for i, v in enumerate(['tic', 'tac', 'toe']):  
...     print i, v  
...  
0 tic  
1 tac  
2 toe
```

2.11. Módulos

Si sales del intérprete de Python y entras de nuevo, las definiciones que has hecho (funciones y variables) se pierden. Por lo tanto, si desea escribir un programa un poco más largo, es mejor usar un editor de texto para preparar la entrada del intérprete y ejecutarlo con ese archivo como entrada.

Un módulo es un archivo que contiene las definiciones y declaraciones de Python. El nombre del archivo es el nombre del módulo con el sufijo `.py`. Por ejemplo, use su editor de texto favorito para crear un archivo llamado `fibonacci.py` en el directorio actual con el siguiente contenido:

```
def fib(n): # escribir la serie Fibonacci hasta n  
    a, b = 0, 1  
    while b < n:  
        print b,  
        a, b = b, a+b  
def fib2(n): # retorne la serie de Fibonacci hasta n  
    result = []  
    a, b = 0, 1  
    while b < n:  
        result.append(b)  
        a, b = b, a+b  
    return result
```

A continuación, introduzca el intérprete de Python e importe este módulo con el siguiente comando:

```
>>> import fibo
```

2.12. Clases

Mecanismo de la clase de Python, añade clases al lenguaje con un mínimo de nueva sintaxis y la semántica.

Las características más importantes de las clases se mantienen con pleno poder, sin embargo: el mecanismo de herencia de clases permite múltiples clases, una clase derivada puede sobre escribir cualquier método de la clase base o de clases, y un método puede llamar al método de la clase base con el mismo nombre. Los objetos pueden contener una cantidad arbitraria de datos.

Sintaxis de definición de clase

La forma más simple de definición de una clase es así:

```
class nombrecase:
```

Las definiciones de clases, como las definiciones de funciones (sentencias def) deben ser ejecutados antes de que tengan algún efecto. (Es posible que usted podría colocar una definición de clase en una rama de una sentencia if, o dentro de una función.)

Clases de objetos.

Clases de objetos soportan dos tipos de operaciones: referencia a atributos e instanciación.

Referencias a atributos utilizan la sintaxis estándar que se utiliza para todas las referencias a atributos en Python: obj.name. Los nombres de atributos válidos son todos los nombres que estaban en espacio de nombres de la clase cuando se creó la clase de objeto. Por lo tanto, la definición tenía este aspecto:

```
class MiClase:
    """Ejemplo de una clase simple"""
    i = 12345
    def f(self):
        return 'Hola Mundo'
```

Entonces MiClase.i y MiClase.f son referencias a atributos válidas, que devuelven un entero y un objeto de función, respectivamente.

Los atributos de clase también se pueden asignar, por lo que puede cambiar el valor de MiClase.i por cesión.

El método de la función se declara con un primer argumento explícito que representa el objeto, que se proporciona implícitamente por la llamada.

A menudo, el primer argumento de un método se llama uno mismo. Esto no es más que una convención: el nombre de uno mismo no tiene absolutamente ningún significado especial para Python.

La instanciación de clases utiliza la notación de funciones. Sólo pretende que el objeto de la clase es una función sin parámetros que devuelve una nueva instancia de la clase. Por ejemplo (suponiendo que la clase anterior):

```
x = MiClase()
```

Crea una nueva instancia de la clase y le asigna este objeto a la variable local x.

Las únicas operaciones comprendidas por los objetos de instancia son referencias a atributos. Hay dos tipos de nombres de atributos válidos, atributos de datos y métodos.

Los atributos de los datos no necesitan ser declarados, al igual que las variables locales, que cobran existencia cuando son asignados primero.

Por ejemplo, si x es la instancia de MiClase creada anteriormente, el siguiente fragmento de código se imprime el valor 16, sin dejar rastro:

```
x.counter = 1
while x.counter < 10:
    x.counter = x.counter * 2
print x.counter
del x.counter
```

Por lo general, un método que se llama justo después de que estar vinculado:
`x.f ()`

En el ejemplo `MiClase`, esto devuelve la cadena 'hola mundo'. Sin embargo, no es necesario llamar a un método de inmediato: `xf` es un objeto método, y puede ser almacenado lejos y llama en un momento posterior. Por ejemplo:

```
xf = x.f
while True:
    print xf()
```

Imprimirá hola mundo hasta el final de los tiempos.

¿Qué sucede exactamente cuando se llama a un método? Lo especial de los métodos es que el objeto se pasa como primer argumento de la función. En nuestro ejemplo, la llamada `xf ()` es exactamente equivalente a `MiClase.f (x)`.

En general, llamar a un método con una lista de n argumentos es equivalente a llamar la función correspondiente con una lista de argumentos que se crea mediante la inserción de objetos del método antes del primer argumento.

Herencia

Por supuesto, una característica del lenguaje no sería digna de ese nombre "clase" sin el apoyo de la herencia. La sintaxis de una definición de clase derivada tiene este aspecto:

El nombre *nombreClaseBase* debe estar definido en un ámbito que contiene la definición de clase derivada. En lugar de un nombre de la clase base, también se permiten otras expresiones arbitrarias. Esto puede ser útil, por ejemplo, cuando la clase base está definida en otro módulo:

```
class NombreClaseBase (modname.NombreClaseBase ):
```


3. Configuración

Este documento contiene soluciones paso a paso para el ejercicio correspondiente folleto que acompaña OpenERP capacitaciones técnicas.

Cada sección ofrece la solución a los ejercicios de la sección correspondiente en el manual de ejercicios. Las instrucciones se darán paso a paso para dar la mayor sencillez posible, por lo general proporcionando un ejemplo de código fuente que permita implementar el comportamiento deseado.

1.1. Open Source RAD con OpenObject

OpenERP es un moderno software de gestión empresarial, publicado bajo la licencia AGPL, y con CRM, RRHH, Ventas, Contabilidad, Producción, Inventario, Gestión de Proyectos, entre otros. Se basa en OpenObject, un sistema modular, marco escalable e intuitivo Rapid Application Development (RAD) escrito en Python.

- **OpenObject** cuenta con una caja de herramientas completa y modulares para la construcción rápida de aplicaciones: Mapeo objeto-relación integrado (ORM) de apoyo, Modelo-Vista-Controlador basado en plantillas (MVC), interfaces de un sistema de generación de informes, automatizada internacionalización, y mucho más.
- **Python** es un lenguaje de programación dinámico de alto nivel, ideal para RAD, combinando potencia con una sintaxis clara, y un núcleo mantiene pequeña por diseño.

Consejos:

Enlaces de interés

- **Sitio web principal, con OpenERP Descargas:** www.openerp.com
- **Documentación funcional y técnica:** [doc.openerp.com / trunk](http://doc.openerp.com/trunk)
- **Recursos de la comunidad:** [www.launchpad.net / OpenObject](http://www.launchpad.net/OpenObject)
- **Servidor de Integración:** demo.openerp.com
- **OpenERP plataforma E-Learning:** edu.openerp.com
- **Python doc:** [docs.python.org/2.6 /](http://docs.python.org/2.6/)
- **PostgreSQL doc:** www.postgresql.org/docs/8.4/static/index.html Python es un lenguaje de alto nivel de programación dinámica, ideal para RAD, poder combinar con una sintaxis clara, y un núcleo mantiene pequeña por diseño.

1.2. Instalación de OpenERP

OpenERP se distribuye como paquetes / instaladores para la mayoría de plataformas, pero por supuesto puede ser instalado desde la fuente en cualquier plataforma.

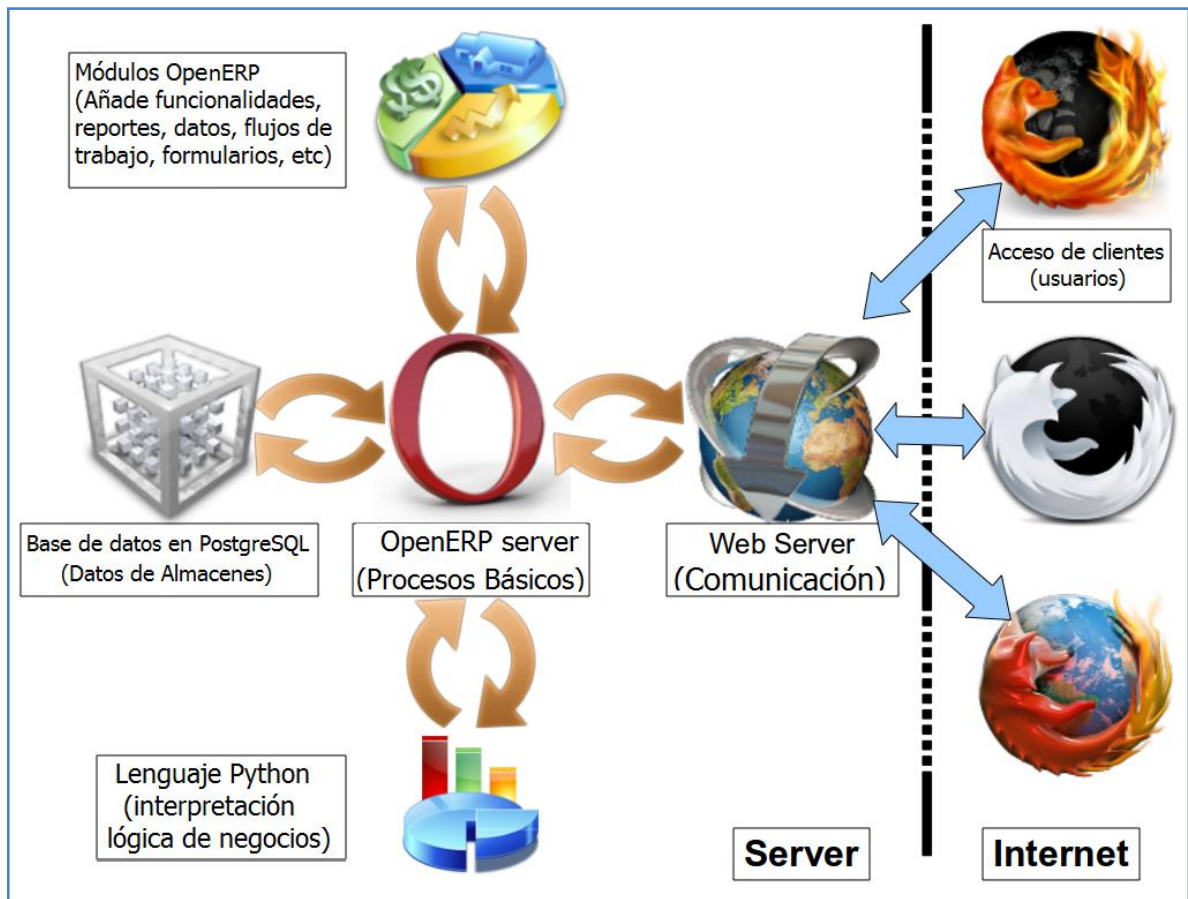
Nota:

El procedimiento de instalación de OpenERP es probable que evolucione (dependencias y demás), así que asegúrese de comprobar siempre la documentación específica (envasado y en la página web) de los últimos procedimientos. Ver <http://doc.openerp.com/install>.

OpenERP Arquitectura

OpenERP utiliza el paradigma cliente-servidor, con diferentes piezas de software que actúan como cliente y el servidor en función de la configuración deseada.

OpenERP proporciona una interfaz web accesible con cualquier navegador moderno.



1.3. Instalación de paquetes

Windows	<i>Instalador todo en uno, y los instaladores independientes para servidor, cliente y servidor web se encuentran en la página web.</i>
Linux	<i>OpenERP-server y paquetes OpenERP-cliente están disponibles a través del correspondiente gestor de paquetes (por ejemplo, Synaptic en Ubuntu) o utilizando Bazar <code>bzr rama lp: OpenERP</code> (o <code>OpenERP / trunk</code> para la versión de tronco) que identifica en Launchpad, entonces <code>cd openerp</code> (<code>cd tronco</code> la versión tronco) y. <code>/ bzr_set.py</code></i>

1.4. Instalación desde el código fuente

Hay dos alternativas:

1. Utilizando un archivo .tar proporcionada en el sitio web.
2. Obtener directamente la fuente usando Bazar (distribuido de control de versiones de fuente).

También es necesario para instalar las dependencias necesarias (PostgreSQL y algunas bibliotecas de Python - véase la documentación doc.openerp.com).

Nota:

OpenERP está basado en Python, no se necesita ningún paso de compilación.

Procedimiento de verificación Bazar típica (desde la consola)

```
$ sudo apt-get install bzip2 # instalar el control de versiones bazar
$ bzip2 branch lp:openerp-tools # obtener instalador de código
$ sh setup.sh # buscar código y realizar la configuración
```

1.5. Creación de base de datos

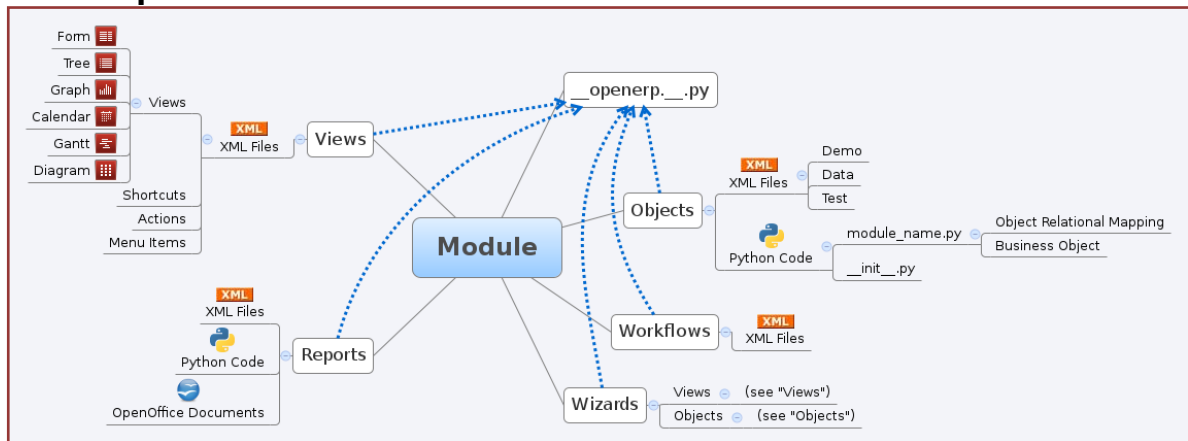
Después de la instalación, ejecutar el servidor y el cliente. En la pantalla de inicio de sesión del cliente Web, haga clic en Administrar bases de datos para crear una nueva base de datos (por defecto la contraseña de súper administrador es **admin**). Cada base de datos tiene sus propios módulos y la configuración.

Nota:

Demostración de que los datos también pueden ser incluidos.

4. Construir un módulo de OpenERP

4.1. Composición de un Módulo



Un módulo puede contener los siguientes elementos:

- **Objeto de negocio:** declara como clases de Python que se extienden de la clase osv.Model OpenObject, la persistencia de estos recursos es manejada completamente por OpenObject.
- **Datos:** Archivos XML / CSV con los metadatos (vistas y la declaración de flujos de trabajo), los datos de configuración (parametrización de módulos) y los datos de demostración (opcional recomendada para pruebas).
- **Wizards:** formularios interactivos con estado utilizados para ayudar a los usuarios, a menudo disponible como acciones contextuales sobre los recursos.
- **Reportes:** RML (formato XML). MAKO u OpenOffice plantillas de informes, que se fusionan con cualquier tipo de datos de negocios y generar HTML, ODT o informes en PDF.

4.2. Estructura del módulo

Cada módulo está contenido en su propio directorio en el servidor de / bin / addons o en otro directorio de complementos, configurado en la instalación del servidor.

Nota:

De forma predeterminada, el único directorio de complementos conocidos por el servidor es el servidor de / bin / addons. Es posible agregar nuevos complementos por:

1) *Copia en addons server / bin /, o crear un enlace simbólico a cada uno de ellos en este directorio.*

2) *Especificando otro directorio que contiene complementos para el servidor. Este último se puede lograr ya sea ejecutando el servidor con la opción-complementos-path = opción, o bien mediante la configuración de esta opción en el archivo openerp_serverrc, generado automáticamente bajo Linux en el directorio principal del servidor cuando se ejecuta con la opción de ahorro. Puede proporcionar varios complementos a la opción addons_path =, separándolas con comas.*

El archivo `__init__.py` es el descriptor de módulo de Python, porque un módulo de OpenERP es también un módulo regular de Python.

Contiene la instrucción de importación aplicado a todos los archivos de Python del módulo, sin la extensión `.py`. Por ejemplo, si un módulo contiene un único archivo llamado python `mimodulo.py`:

```
import mimodulo
```

El archivo `__openerp__.py` es realmente la declaración del módulo de OpenERP. Es obligatorio en cada módulo. Contiene un único diccionario python con varias piezas importantes de información, como el nombre del módulo, su descripción, la lista de los otros módulos de OpenERP la instalación de que se requiere para el módulo actual para que funcione correctamente. También contiene, entre otras cosas, una referencia a todos los archivos de datos (xml, csv, yml ...) del módulo. Su estructura general es la siguiente (consulte la documentación oficial para una descripción completa del archivo):

```
{
    "name": "MiModulo",
    "version": "1.0",
    "depends": ["base"],
    "author": "Nombre del Autor",
    "category": "Categoria",
    "description": """\
        Description del modulo
        """,
    'data': [
        'mymodule_view.xml', # Archivos de datos, (excepto los datos de demostración y pruebas)
    ],
    'demo': [
        # Archivos que contienen datos de demostración
    ],
    'test': [
        # Archivos que contienen las pruebas
    ],
    'installable': True,
    'auto_install': False,
}
```

Ejercicio 1 - Creación del Módulo

Creación de la Academia Abierta módulo vacío, con un archivo `__openerp__.py`
Realice la instalación de acuerdo con OpenERP.

- Cree una nueva carpeta "AcademiaAbierta" en algún lugar en el equipo.
- Crear una "`__init__.py`" vacía
- Crear un archivo "`__openerp__.py`" bajo el proyecto " AcademiaAbierta ":

```
{
    "name": "Academia Abierta",
    "version": "1.0",
    "depends": ["base"],
    "author": "Yo",
    "category": "Prueba",
    "description": """\
        Módulo Academia abierta para la gestión de cursos de formación:
        - Cursos de formación
        - Sesiones de formación
        - Registro de asistenteso
        """,
    'data': [
    ],
}
```

4.3. Object Service – ORM

Componente clave de OpenObject, el **Object Service** (OSV) implementa una capa completa de mapeo objeto-relacion, liberando a los desarrolladores de tener que escribir la plomería SQL básica. Los objetos de negocio se declaran como clases Python que heredan de la clase `osv.Model`, hacen que sean parte del Modelo OpenObject, y mágicamente persistieron por la capa ORM.

Atributos `osv.Model` predefinidas para objetos de negocio

<code>_name</code> (requerido)	Nombre del objeto, en la notación de puntos (en el módulo de espacio de nombres)
<code>_columns</code> (requerido)	diccionario {nombres de los campo > declaración de los campos}
<code>_defaults</code>	Diccionario: {nombres de campo> Funciones que proporcionan valores por defecto} <code>_defaults ['nombre'] = lambda self, cr, uid, el contexto: 'carro'</code>
<code>_rec_name</code>	Campo alternativo para usar como nombre, utilizado por <code>name_get</code> de <code>osv ()</code> (por defecto: "nombre")

4.4. Tipos de campos ORM

Los objetos pueden contener tres tipos de campos: simples, relacional y funcional. Los tipos simples son enteros, reales, booleanos, cadenas, etc. Los campos relacionales representan las relaciones entre objetos (`one2many`, `many2one`, `many2many`). Los funcionales no se almacenan en la base de datos, pero calculan sobre la marcha como las funciones de Python.

Atributos comunes soportados por todos los campos (opcionales a menos que se especifique).

- **string:** Etiqueta Campo (requerido).
- **required:** True si es obligatorios.
- **readonly:** True si no es editable.
- **help:** Ayuda sobre herramientas.
- **select:** 1 para incluir en las vistas de búsqueda y optimización para el filtrado de lista (con un índice de base de datos) Nombre del objeto de negocio, en la notación de puntos (en el módulo de espacio de nombres)
- **context:** Diccionario con parámetros contextuales (para campos relacionales)

Campos simples

- **boolean(...)** **integer(...)** **date(...)** **datetime(...)** **time(...)**

```
'fecha_de_inicio': fields.date('Fecha de inicio')
'activo': fields.boolean('Activo')
'prioridad': fields.integer('Prioridad')
```

- **char(string, size, translate=False,...)** **text(string, translate=False,...)**
 - ✓ **translate:** *True* si los valores del campo se pueden traducir por los usuarios
 - ✓ **size:** tamaño máximo para los campos CHAR (41,45!)
- **float(string, digits=None, ...)**
digits: tuple (precisión, escala). Si no se proporcionan cifras, es un real, no un tipo decimal.

4.5. Especial/Nombres de los campos reservados

Algunos nombres de campo están reservados para el comportamiento predefinido en OpenObject. Algunos de ellos son creados automáticamente por el sistema, y en ese caso se ignorará cualquier campo con ese nombre.

<i>id</i>	<i>identificador de sistema único para el objeto (creado por ORM, no lo agregue)</i>
<i>name</i>	<i>define el valor que se utiliza de forma predeterminada para mostrar el registro en las listas, etc, si falta, ajuste _rec_name para especificar otro campo que se utilizará para este fin</i>

Ejercicio 2 - Definir un modelo

Definir un nuevo modelo de datos llamado `Curso` en el módulo `AcademiaAbierta`. El curso tiene un campo nombre, o "título", y un campo descripción. Se requiere el nombre de la clase.

- a) En el marco del nuevo proyecto "AcademiaAbierta", cree un nuevo archivo "curso.py":

```
from openerp.osv import osv, fields

class curso(osv.Model):
    _name = "academiaabierta.curso" #Nombre de la clase

    _columns = {
        #Campos de la clase
        'nombre' : fields.char(string="Título", size=256, required=True),
        'descripcion' : fields.text(string="Descripción"),
    }
```

- b) Crear un archivo "__init__.py" dentro de la carpeta " AcademiaAbierta" y escribir (para así llamar al modulo curso)

```
import curso
```

4.6. Acciones y menús

Las acciones son declarados como registros periódicos y se pueden activar de tres formas:

- Haciendo clic en los botones en las vistas, si éstas están conectadas a las acciones
- Haciendo clic en los elementos de menú vinculado a una acción específica.
- Como acciones contextuales en un objeto

Los menús también son registros regulares en el ORM. Sin embargo, se declaran con la `<menuitem>` tag. Este es un atajo para la declaración de un registro "ir.ui.menu", y conéctelo con una acción correspondiente a través de un registro "ir.model.data".

El siguiente ejemplo define un elemento de menú para mostrar la lista de ideas. La acción asociada al menú menciona el modelo de los registros para mostrar, y vistas que están habilitados; en este ejemplo, sólo las vistas de árbol y la forma estará disponible. Existen otros campos opcionales para las acciones, consulte la documentación para obtener una descripción completa de los mismos.

```
<record model="ir.actions.act_window" id="action_list_ideas">
  <field name="name">Ideas</field>
  <field name="res_model">idea.idea</field>
  <field name="view_mode">tree,form</field>
  <field name="help" type="html">
    <p class="oe_view_nocontent_create"> Realizar registro</p>
  </field>
</record>

<menuitem id="menu_ideas" parent="menu_root" name="Ideas" action="action_list_ideas"/>
```


Nota:

La acción la presentamos en general antes de su menú correspondiente en el archivo XML. Esto se debe a que el disco "action_id" debe existir en la base de datos para permitir que el ORM pueda crear el registro del menú.

Ejercicio 3 - Definir nuevas entradas de menú

Definir nuevas entradas de menú para acceder a los cursos y sesiones en virtud de la entrada del menú OpenAcademy; uno debe ser capaz de: 1) mostrar una lista de todos los cursos v 2) crear / modificar nuevos cursos

1. Crear en la carpeta **view/** el archivo **academiaabierta.xml**:

```
<?xml version="1.0" encoding="UTF-8"?>
<openerp>
  <!-- Asi iniciamos un nuevo comentario y así lo terminamos -->
  <!-- Todos los datos en OpenERP se almacenan dentro / OpenERP / rama de datos
       que significa en XML:
       <openerp>
         <data>
         </data>
       </openerp>
  -->
  <!--! Sangría en XML es opcional pero por favor trate de permanecer legible -->
  <data>
    <!-- Ventanas de acción -->
    <!-- La etiqueta siguiente es una definición de acción.
         Básicamente, creamos un registro en el model ir.actions.act_window y
         OpenERP hará el resto -->
    <record model="ir.actions.act_window" id="curso_list_action">
      <field name="name">Curso</field>
      <field name="res_model">academiaabierta.curso</field>
      <field name="view_mode">tree,form</field>
      <field name="help" type="html">
        <p class="oe_view_nocontent_create"> Realizar registro</p>
      </field>
    </record>

    <!-- menuitems -->
    <menuitem id="inicio_academiaabierta_menu" name="Academia Abierta" />
    <!-- Se necesita un primer nivel en el menú de la izquierda
         antes de usar el atributo action = -->
    <menuitem id="academiaabierta_menu" name="Academia Abierta"
      parent="inicio_academiaabierta_menu" />
    <!-- Lo siguiente menuitem debe aparecer * después * de su
         openacademy_menu padre y * después * de su curso_list_action acción -->
    <menuitem id="curso_menu" name="Cursos" parent="academiaabierta_menu"
      action="curso_list_action" />
    <!-- Identificación completa de ubicación: action = "academiaabierta.curso_list_action"
         No se requiere cuando se trata del mismo módulo -->
  </data>
</openerp>
```


2. In "__openerp__.py":

```
{
    'name' : "Academia Abierta",
    (...)
    'data' : [
        'view/academiaabierta.xml',
    ],
}
```

5. Construcción de vistas: conceptos básicos

Vistas forman una jerarquía. Varios puntos de vista del mismo tipo se pueden declarar en el mismo objeto, y se utilizarán en función de sus prioridades.

También es posible añadir / eliminar elementos en una vista al declarar una vista heredada (ver Ver la herencia).

1.1. Declaración de vista genérica

Una vista se declara como un registro del modelo ir.ui.view. Dicho registro se declara en XML como:

```
<record model="ir.ui.view" id="view_id">
  <field name="name">nombre.vista</field>
  <field name="model">nombre_objeto</field>
  <field name="priority" eval="16"/>
  <field name="arch" type="xml">
    <!-- view content: <form> formulario,
      <tree> lista, <calendar> calendario,
      ect... -->
  </field>
</record>
```

Observe que el contenido de la vista en sí se define como XML. Por lo tanto, el arco de campo del registro debe ser declarado de tipo XML, con el fin de analizar el contenido de ese elemento como el valor del campo.

1.2. Vista Tree

Vistas Tree, también llamadas vistas de lista, los registros se visualizan en forma de tabla. Se definen con el elemento XML <tree>. En su forma más simple, menciona los campos que deben ser utilizados como las columnas de la tabla.

```
<field name="arch" type="xml">
  <tree string="Lista de Ideas">
    <field name="nombre"/>
    <field name="inventor_id"/>
  </tree>
</field>
```

1.3. Vista Form

Los formularios permiten la creación / edición de recursos, y se definen por elementos XML <form>. El siguiente ejemplo muestra como una vista de formulario puede ser espacialmente estructurada con separadores, grupos, pestañas, etc Consulte la documentación para saber todos los posibles elementos que se pueden colocar en un formulario.

```
<field name="arch" type="xml">
  <form string="Formulario Ideas">
    <group colspan="2" col="2">
      <separator string="Material General" colspan="2"/>
      <field name="nombre"/>
      <field name="inventor_id"/>
    </group>
    <group colspan="2" col="2">
      <separator string="Fecha" colspan="2"/>
      <field name="activo"/>
      <field name="fecha_invento" readonly="1"/>
    </group>
    <notebook colspan="4">
      <page string="Descripcion">
        <field name="descripcion" nolabel="1"/>
      </page>
    </notebook>
    <field name="estado"/>
  </form>
</field>
```

Ahora, con la nueva versión 7 se puede escribir HTML en tu formulario.

```
<field name="arch" type="xml">
  <form string="Idea Form v7" version="7.0">
    <header>
      <button string="Confirm" type="object" name="action_confirm"
        states="draft" class="oe_highlight" />
      <button string="Mark as done" type="object" name="action_done"
        states="confirmed" class="oe_highlight"/>
      <button string="Reset to draft" type="object"
        name="action_draft" states="confirmed,done" />
      <field name="state" widget="statusbar"/>
    </header>
    <sheet>
      <div class="oe_title">
        <label for="name" class="oe_edit_only" string="Nombre de Idea" />
        <h1><field name="name" /></h1>
      </div>
      <separator string="General" colspan="2" />
      <group colspan="2" col="2">
        <field name="description" placeholder="Idea descripcion..." />
      </group>
    </sheet>
  </form>
</field>
```

Ejercicio 1 - Personalice el formulario utilizando XML

Crea tu propia vista de formulario para el objeto del curso. Los datos mostrados deben ser: el nombre y la descripción del curso.

1. En academiaabierta _view.xml cree las siguientes vistas:

```
<record model="ir.ui.view" id="curso_form_view">
  <field name="name">curso.form</field>
  <field name="model">academiaabierta.curso</field>
  <field name="arch" type="xml">
    <form string="Curso Form">
      <field name="nombre" />
      <field name="descripcion" />
    </form>
  </field>
</record>
```

Nota:

Por defecto, la vista se separa en 4 columnas

2. Podemos mejorar la vista formulario para una mejor visualización.

```
<record model="ir.ui.view" id="curso_form_view">
  <field name="name">curso.form</field>
  <field name="model">academiaabierta.curso</field>
  <field name="arch" type="xml">
    <form string="Curso Form">
      <!-- Nota: las columnas del form por defecto son 4,
      la definición de un colspan de 4 estirará el campo
      en el formulario entero -->
      <field name="nombre" colspan="4" />
      <field name="descripcion" colspan="4" />
    </form>
  </field>
</record>
```

Ejercicio 2 - Pestañas

En la vista de formulario de curso, poner el campo de descripción en una pestaña, de tal manera que será más fácil añadir otras fichas después, que contiene información adicional.

Modificar la vista de formulario Curso de la siguiente manera:

```
<record model="ir.ui.view" id="curso_form_view">
  <field name="name">curso.form</field>
  <field name="model">academiaabierta.curso</field>
  <field name="arch" type="xml">
    <form string="Curso Form">
      <field name="nombre" colspan="4" />
      <notebook colspan="4">
        <page string="Descripción">
          <field name="descripcion" colspan="4" nolabel="1" />
        </page>
        <page string="About">
          <label string="Este es un ejemplo de pestañas" />
        </page>
      </notebook>
    </form>
  </field>
</record>
```

6. Relaciones entre Objetos

Ejercicio 1 - Crear clases

Crear las clases de sesión y de asistentes, y agregar un elemento de menú y una acción para mostrar las sesiones. Una sesión tiene un nombre (requerido), una fecha de inicio, el tiempo (en días) y un número de escaños. Un participante tiene un nombre, que no es necesario.

1. Crear clases sesion y de asistentes.

```
class Sesion(osv.Model):
    _name = "academiaabierta.sesion"
    _columns = {
        'nombre' : fields.char(string="Nombre", size=100, required=True),
        'fecha_inicio' : fields.date(string="Fecha de Inicio"),
        'duracion' : fields.float(string="Duracion", digits=(6,2),
            help="Duracion en dias"),
        'sillas' : fields.integer(string="Numero de sillas"),
    }

class Asistentes(osv.Model):
    _name = "academiaabierta.asistentes"
    _columns = {
        'name' : fields.char(string="Nombre", size=100),
    }
```

Nota:

digits=(6,2) especifica la precisión de un número float: 6 es el número total de dígitos, mientras que el 2 es el número de dígitos después de la coma. Tenga en cuenta que el resultado es que el número de dígitos antes de la coma es máxima 4 en este caso.

1.1. Campos relacionales

Campos relacionales son campos con los valores que se refieren a otros objetos.

- Atributos comunes soportados por campos relacionales
 - domain: restricción opcional en forma de argumentos para la búsqueda (ver search ())
- **many2one (obj, ondelete = 'set null', ...):** relación simple hacia otro objeto (el uso de una clave externa)

Estado

Codigo de Ciudad



Amazonas
Aragua
Carabobo
Falcon
Guarico
Merida
Zulia
[Crear y editar...](#)

- obj: nombre del objeto de destino (*obligatorio*)
- ondelete: manejo eliminación: 'set null', 'cascada', 'restringir' (consulte la documentación de PostgreSQL).

- **one2many (obj, field_id, ...):** relación virtual a varios objetos (inversas de many2one). Un campo relacional one2many proporciona un tipo de relación "contenedor": los registros del otro lado de la relación puede ser visto como "contenido" en el registro de este lado.

Producto	Cantidad
<div></div>	0,000 <div></div>
Añadir un elemento	

- obj: nombre del objeto de destino (*obligatorio*)
- field_id: nombre del campo inverso de many2one, es decir, la clave externa correspondiente (requerido)

- **many2many (obj, rel, campo1, campo2, ...):** relación bidireccional múltiple entre objetos. Este es el tipo más general de la relación: un registro puede estar relacionado con cualquier número de registros en el otro lado, y vice-versa.

+

 Add

—

 Remove

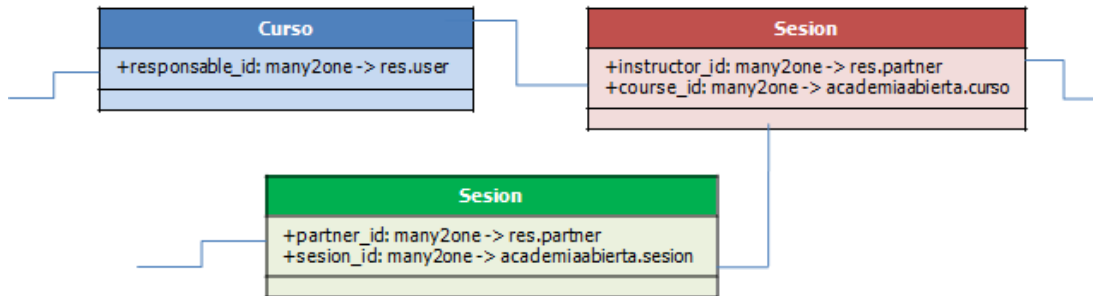
Name	City	Country	Type	Code	# of Contact

- obj: nombre del objeto de destino (*obligatorio*)
- rel: tabla de relaciones de utilizar
- campo1: nombre del campo en la tabla rel almacenar el id del objeto actual
- campo2: nombre del campo en la tabla rel almacenar el id del objeto de destino

Tenga en cuenta que los parámetros rel, campo1 y campo2 son opcionales. Cuando no se les da, el ORM genera automáticamente los valores adecuados para ellos.

Ejercicio 2 - Crear clases

Crear las clases de sesión y de asistentes, y agregar un elemento de menú y una acción para mostrar las sesiones. Una sesión tiene un nombre (requerido), una fecha de inicio, el tiempo (en días) y un número de escaños. Un participante tiene un nombre, que no es necesario.



1. Modificar las clases de la siguiente manera.

```
class Curso(osv.Model):
    _name = "academiaabierta.curso"
    _columns = {
        'nombre' : fields.char(string="Titulo", size=100, required=True),
        'descripcion' : fields.text(string="Descripcion"),
        #Campo relacional
        'responsable_id' : fields.many2one('res.user', ondelete='set null',
            string="Titulo", select=True),
        # 'Set null' se restablecerá responsable_id a
        # Indefinido si ha sido eliminada responsable
    }
```

```
class Sesion(osv.Model):
    _name = "academiaabierta.sesion"
    _columns = {
        'nombre' : fields.char(string="Nombre", size=100, required=True),
        'fecha_inicio' : fields.date(string="Fecha de Inicio"),
        'duracion' : fields.float(string="Duracion", digits=(6,2),
            help="Duracion en dias"),
        'sillas' : fields.integer(string="Numero de sillas"),
        #Campo relacional
        'instructor_id' : fields.many2one('res.partner', string="Instructor"),
        'curso_id' : fields.many2one('academiaabierta.curso',
            ondelete='cascade', string="Curso", required=True),
        # 'cascade' destruirá la sesión si se ha eliminado curso_id
    }
```

```
class Asistentes(osv.Model):
    _name = "academiaabierta.asistentes"
    # _rec_name Redefinir el campo para llegar al ver el registro en
    # Otro objeto. En este caso, el nombre de la pareja será impreso
    _rec_name = 'partner_id'
    # (útil cuando no hay campo "nombre" definido)
    _columns = {
        'partner_id': fields.many2one('res.partner', string="Partner",
                                       required=True, ondelete='cascade'),
        'sesion_id': fields.many2one('academiaabierta.sesion', string="Sesion",
                                       required=True, ondelete='cascade'),
    }

```

Nota:

En la clase Asistentes, hemos eliminado el nombre del campo. En lugar de ese campo, el nombre partner va a ser usado para identificar el registro de asistentes. Esto es para lo que se utiliza atributo _rec_name.

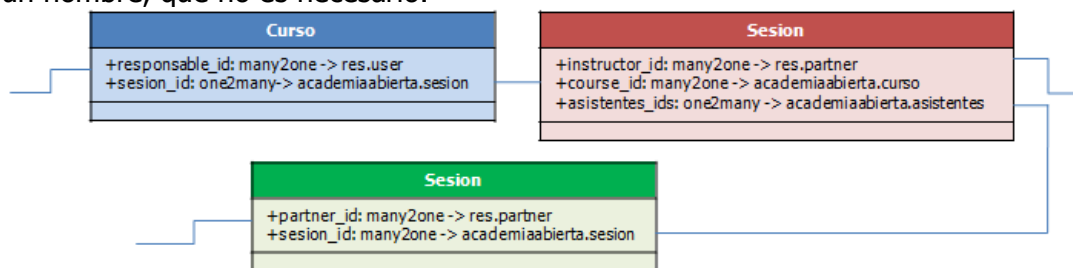
2. Añadir el acceso al objeto de sesión en 'view / academiaabierta.xml'

```
<record model="ir.actions.act_window" id="sesion_lista_accion">
    <field name="name">Sesion</field>
    <field name="model">academiaabierta.sesion</field>
    <field name="view_type">form</field>
    <field name="view_mode">tree, form</field>
</record>
<menuitem id="sesion_menu" name="Sesiones" parent="academiaabierta_menu"
action="sesion_lista_accion" />

```

Ejercicio 3 - Crear clases

Crear las clases de sesión y de asistentes, y agregar un elemento de menú y una acción para mostrar las sesiones. Una sesión tiene un nombre (requerido), una fecha de inicio, el tiempo (en días) y un número de escaños. Un participante tiene un nombre, que no es necesario.



1. Modificar las clases de la siguiente manera.


```

class Curso(osv.Model):
    _name = "academiaabierta.curso"
    _columns = {
        'nombre' : fields.char(string="Titulo", size=100, required=True),
        'descripcion' : fields.text(string="Descripcion"),
        'responsable_id' : fields.many2one('res.user', ondelete='set null',
            string="Titulo", select=True),
        # A one2many es el enlace inverso de un many2one
        'sesion_id': fields.one2many('academiaabierta.sesion', 'curso_id',
            string="Sesion"),
    }

```

```

class Sesion(osv.Model):
    _name = "academiaabierta.sesion"
    _columns = {
        'nombre' : fields.char(string="Nombre", size=100, required=True),
        'fecha_inicio' : fields.date(string="Fecha de Inicio"),
        'duracion' : fields.float(string="Duracion", digits=(6,2),
            help="Duracion en dias"),
        'sillas' : fields.integer(string="Numero de sillas"),
        'instructor_id' : fields.many2one('res.partner', string="Instructor"),
        'curso_id' : fields.many2one('academiaabierta.curso',
            ondelete='cascade', string="Curso", required=True),
        'asistentes_ids' : fields.one2many('academiaabierta.asistentes', 'sesion_id',
            string='Asistentes')
    }

```

Nota:

La definición de un campo *one2many* requiere de el campo *many2one* correspondiente a definirse en la clase a que se refiere.

Este nombre de campo *many2one* es dado como segundo argumento al campo *one2many* (en este caso, *curso_id*).

Ejercicio 4 – Modificas las vistas

Modificar / Crear las vistas tree y form de los objetos Curso y de sesion. Los datos mostrados deben ser:

Para el objeto del curso:

- En la vista tree, el nombre del curso y el responsable de ese curso;
- En la vista form, el nombre y el responsable de la parte superior, así como la descripción del curso en una pestaña, y las sesiones relacionadas en una segunda pestaña.

Para el objeto de la sesion:

- En la vista de árbol, el nombre de la sesión y el curso relacionado.
- En la vista formulario, todos los campos del objeto Sesion.
- Debido a que la cantidad de datos a la pantalla es bastante grande, tratar de organizar las vistas de formulario, de tal manera que la información looks clear.

1. Modificamos la vista Curso

```

<record model="ir.ui.view" id="curso_form_view">
  <field name="name">curso.form</field>
  <field name="model">academiaabierta.curso</field>
  <field name="arch" type="xml">
    <form string="Curso Form">
      <field name="nombre" colspan="4" />
      <field name="responsable_id" />
      <notebook colspan="4">
        <page string="Descripción">
          <field name="descripcion" colspan="4" nolabel="1" />
        </page>
        <page string="About">
          <label string="Este es un ejemplo de pestañas" />
        </page>
        <!-- Vamos a hacer una vista tree dentro de una vista form!
        Sólo necesitamos las etiquetas <tree> y <form> definir qué
        campo que necesitamos -->
        <page string="Sesiones">
          <field name="sesion_ids" nolabel="1" colspan="4" mode="tree">
            <tree string="Registrar Sesiones">
              <field name="nombre"/>
              <field name="instructor_id"/>
            </tree>
          </field>
        </page>
      </notebook>
    </form>
  </field>
</record>

```

```

<!-- La siguiente declaración se sobreponen a la vista de nuestro modelo
openacademy.curso tree por defecto -->
<record model="ir.ui.view" id="curse_tree_view">
  <field name="name">Curso.tree</field>
  <field name="model">academiaabierta.curso</field>
  <field name="arch" type="xml">
    <tree string="Curso Tree">
      <field name="name" />
    </tree>
  </field>
<!-- Recuerde que cualquier etiqueta que no tenga hijos se puede escribir
como la de arriba. Pero todavía se puede escribir el formulario XML
completo a su preferencia.
-->
  <field name="responsable_id" />
</tree>
</field>
</record>

```

Nota:

En el ejemplo anterior, si la vista form no se hubiera personalizado mostraría el campo `session_ids` en vez de sólo el nombre y el instructor, aparecería el campo `curso_id` (así como todos los demás campos del objeto de sesión), de ser requeridos. Sin embargo,

cualquiera que sea el valor se introduce en el campo curso_id, sería ignorada cuando se guarda la instancia de la sesión, y se sustituye por el cursp_id de la instancia actual del objeto del curso.

2. Creamos la vista para sesión.

Nota:

Por defecto, la vista se separa en 4 columnas. Podemos mejorar la vista del formulario de sesión para una mejor visualización mediante el atributo colspan.

```
<record model="ir.ui.view" id="sesion_form_view">
  <field name="name">sesion.form</field>
  <field name="model">academiaabierta.sesion</field>
  <field name="arch" type="xml">
    <form string="sesion Form">
      <group colspan="2" col="2">
        <separator string="General" colspan="2" />
        <field name="curso_id" />
        <field name="nombre" />
        <field name="instructor_id"/>
      </group>
      <group colspan="2" col="2">
        <separator string="Horario" colspan="2" />
        <field name="fecha_inicio" />
        <field name="duracion" />
        <field name="seats" />
      </group>
      <separator string="Asistentes" colspan="4" />
      <field name="asistente_ids" colspan="4" nolabel="1">
        <!-- Atributo 'editable' fijará la posición de cargar
              nuevos elementos en la lista -->
        <tree string="" editable="bottom">
          <field name="partner_id" />
        </tree>
      </field>
    </form>
  </field>
</record>
```

```

<record model="ir.ui.view" id="sesion_tree_view">
  <field name="name">sesion.tree</field>
  <field name="model">academiaabierta.sesion</field>
  <field name="arch" type="xml">
    <tree string="Sesion Tree">
      <field name="nombre"/>
      <field name="curso_id"/>
    </tree>
  </field>
</record>

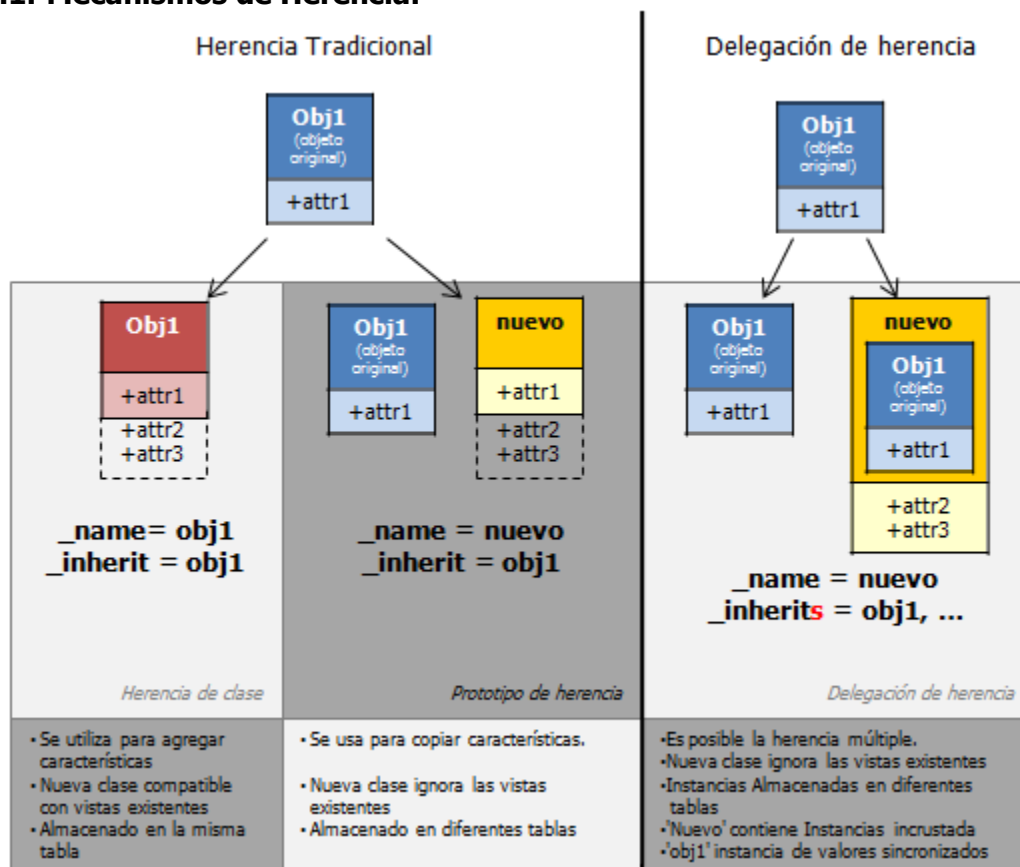
```

Nota:

Aquí los grupos aparecen como dos columnas; podríamos mostrarles en dos líneas estableciendo el "colspan" de los grupos de "4" o utilizando la etiqueta <newline/>.

7. Herencia

7.1. Mecanismos de Herencia.



Predefinidas osv.Model atributos de los objetos de negocio

<u>_inherit</u>	Predefinidos atributos osv.Model de los objetos de negocio.
<u>_inherits</u>	Para varios mecanismos de herencia: el mapeo del diccionario _name que sirve como llave foránea de donde los hijos llaman los atributos del padre.

7.2. Herencia de vista.

Existen vistas que deben ser modificadas a través de herencia de vista, y nunca directamente. Una herencia de vista hace referencia a la vista padre utilizando el campo `inherit_id`, y puede añadir o modificar los elementos existentes en la vista haciendo referencia a ellos a través de los selectores de elementos o expresiones XPath (especificando la posición adecuada).

Posiciones	<ul style="list-style-type: none">• inside (dentro): colocado en el interior del campo (por defecto).• before (antes): se coloca antes del campo.• attributes (atributos): cambiar atributo de la etiqueta.• replace (sustituir): reemplazar campo.• after (después): se coloca tras el campo.
-------------------	---

Nota:

Información sobre XPath se puede encontrar en: www.w3.org/TR/xpath

```
<record id="idea_category_list2" model="ir.ui.view">
  <field name="name">id.category.list2</field>
  <field name="model">ir.ui.view</field>
  <field name="inherit_id" ref="id_category_list"/>
  <field name="arch" type="xml">
    <!-- Nombre del campo a buscar-->
    <!-- Indicamos la posición que ocupara el nuevo campo -->
    <field name="name" position="before">
      <!--Luego colocar el campo nuevo-->
      <field name="reference" />
    </field>
    <!-- Buscar el campo description en el interior la vista tree, y
    se añade el nuevo campo indicando en que posición aparecerá -->
    <xpath expr="/tree/field[@name='description']" position="after">
      <field name="idea_ids" string="Número de ideas"/>
    </xpath>
  </field>
</record>
```

Ejercicio 1 – Añade un mecanismo de herencia

Utilizando la herencia de clases, cree una clase "Socio", que modifica la clase "Partner" existente, añadiendo un campo booleano al que llamaremos "es_instructor", y la lista de las sesiones que Partner va a atender. Usando la herencia de vistas, modificar vista actual de "Partner" para mostrar los nuevos campos.

1. Crear un archivo "socio.py" dentro del proyecto "AcademiaAbierta". Cree la siguiente clase:

```

from openerp.osv import osv, fields
class Socio(osv.Model):
    """Herenciando a res.partner"""
    # La línea de arriba es la manera para documentar en Python
    # Sus objetos (como clases)
    _inherit = 'res.partner'
    _columns = {
        # Agregamos un nuevo campo al modelo res.partner
        'instructor' : fields.boolean("Instructor"),
    }
    _defaults = {
        # Por defecto, ningún socio es un instructor
        'instructor' : False,
    }

```

2. Crear un archivo "vista_socio.xml" bajo el proyecto " AcademiaAbierta". Crea las siguientes vistas heredadas:

Nota:

Esta parte es la oportunidad de entrar en el modo de programador a inspeccionar el objeto de modificar, encontrar el xml_id de la vista y el lugar para poner el nuevo campo. Puede activarlo poniendo "?Debug" antes del "#".

```

<?xml version="1.0" encoding="UTF-8"?>
<openerp>
  <data>
    <!-- Agregar el campo instructor a la vista ya existente -->
    <record model="ir.ui.view" id="partner_instructor_form_view">
      <field name="name">partner.instructor</field>
      <field name="model">res.partner</field>
      <field name="inherit_id" ref="base.view_partner_form" />
      <field name="arch" type="xml">
        <field name="is_company" position="before">
          <field name="instructor" />
          <label for="instructor" string="¿Es un Instructor?" />
        </field>
      </record>

    <record model="ir.actions.act_window" id="contacto_list_action">
      <field name="name">Contactos</field>
      <field name="res_model">res.partner</field>
      <field name="view_type">form</field>
      <field name="view_mode">tree,form</field>
    </record>

    <menuitem id="configuracion_menu" name="Configuración"
      parent="academiaabierta_menu" />
    <menuitem id="contact_menu" name="Contactos"
      parent="configuracion_menu" action="contacto_list_action" />
  </data>
</openerp>

```

Nota:

Los elementos colocados en el interior de un campo de nivel superior que tiene una posición de atributo **position="after"** (después) se mostrarán en la vista después del campo identificado en la vista padre. Si hay más de un elemento, se muestran en orden inverso al de su aparición en el código XML

3. En el archivo "__init__.py" añadir la siguiente línea:

```
import socio
```

4. Abrir el archivo "__openerp__.py" y añadir "**view / socio.xml**" dentro de "datos" para indicar que reconozca ese archivo y la ubicación:

```
'data' : [  
    'view/academiaabierta.xml',  
    'view/socio.xml',  
],
```

Domains

Los domains se utilizan para seleccionar un subconjunto de los registros de un modelo, de acuerdo con una lista de criterios. Cada criterio es una tupla de tres elementos: un nombre de campo, un operador (véase el manual técnico para más detalles), y un valor. Por ejemplo, el siguiente domain selecciona todos los productos de tipo de servicio que tienen un precio_unit mayor que 1000.

```
[('tipo_producto', '=', 'servicio'), ('precio_unit', '>', 1000)]
```

Por defecto, los criterios se combinan con el operador lógico "AND" Uno puede insertar los operadores lógicos '&', '|', (Respectivamente AND, OR, NOT) en el interior de los domains '!'. Ellos se introducen usando una notación de prefijo, es decir, el operador se pone antes de sus argumentos. El siguiente ejemplo codifica la condición: "estar al servicio de tipo o no tener un precio unitario entre 1000 y 2000".

```
[ '|', ('tipo_producto', '=', 'servicio'),  
  '!', '&', ('precio_unit', '>=', '1000'), ('precio_unit', '<', '2000')]
```

Ejercicio 4 – Domain

Añade un mecanismo que permita a la vista de formulario de sesiones, que el usuario elija el instructor sólo entre los socios del campo "Instructor" de la cual se establece "True".

1. Modifique la clase Sesion de la siguiente forma:

```
class Sesion(osv.Model):  
    _name = "academiaabierta.sesion"  
    _columns = {  
        (...)  
        'instructor_id' : fields.many2one('res.partner', string="Instructor",  
            domain=[('instructor','=',True)]),  
        (...)  
    }
```

7.3.

7.4.

7.5.

7.6.

8.

9.

10.

11.

12.

13.