



## 5 Desenvolvimento de componentes

## Sumario

|                                                                                               |    |
|-----------------------------------------------------------------------------------------------|----|
| 5 Desenvolvemento de compoñentes.....                                                         | 1  |
| 5.1 Convencións empregadas.....                                                               | 5  |
| 5.2 Técnicas e estándares. Modelo-Vista-Controlador.....                                      | 6  |
| 5.2.1 Especificacións técnicas para o desenvolvemento de compoñentes.....                     | 7  |
| 5.2.2 Especificacións funcionais para o desenvolvemento de compoñentes.....                   | 8  |
| 5.3 Técnicas de optimización de consultas e acceso a grandes volumes de información.....      | 10 |
| 5.3.1 Operacións de consulta. Ferramentas.....                                                | 10 |
| 5.3.2 Sistemas batch inputs. Xeración de programas de extracción e procesamento de datos..... | 11 |
| 5.4 Linguaxe proporcionada polos sistemas ERP-CRM.....                                        | 12 |
| 5.4.1 Características e sintaxes da linguaxe.....                                             | 12 |
| 5.4.2 Declaración de datos. Tipos básicos.....                                                | 15 |
| 5.4.3 Estruturas de programación. Coleccións.....                                             | 16 |
| 5.4.4 Sentenzas da linguaxe.....                                                              | 17 |
| 5.4.5 Chamadas a funcións.....                                                                | 18 |
| 5.4.6 Clases e obxectos.....                                                                  | 19 |
| 5.4.7 Módulos e paquetes.....                                                                 | 20 |
| 5.4.8 Librerías de funcións (APIs).....                                                       | 20 |
| 5.4.9 Inserción, modificación e eliminación de datos en obxectos.....                         | 21 |
| 5.5 Contornas de desenvolvemento e ferramentas de desenvolvemento en sistemas ERP-CRM.....    | 22 |
| 5.5.1 Depuración dun programa.....                                                            | 22 |
| 5.5.2 Manexo de erros.....                                                                    | 24 |
| 5.6 Formularios e informes nun sistemas ERP-CRM.....                                          | 25 |
| 5.6.1 Arquitectura de formularios e informes. Elementos.....                                  | 25 |
| 5.6.2 Ferramentas para a creación de formularios e informes.....                              | 27 |

## Índice de figuras










|                                                                     |    |
|---------------------------------------------------------------------|----|
| Figura 1: Modelo-Vista-Controlador.....                             | 6  |
| Figura 2: Arquitectura cliente-servidor.....                        | 7  |
| Figura 3: Módulo basee.....                                         | 9  |
| Figura 4: Guido van Rossum.....                                     | 13 |
| Figura 5: Terminal de ArcoLinux ejecutando Ola Mundo en Python..... | 14 |
| Figura 6: Terminal de ArcoLinux ejecutando un programa Python.....  | 14 |
| Figura 7: Depurador IDLE para a linguaxe Python en Macintosh.....   | 23 |



Material docente elaborado a partir e a base dos materiais formativos de FP en liña propiedade do Ministerio de Educación e Formación Profesional.

[Aviso Legal](#)

## 5.1 Convencións empregadas

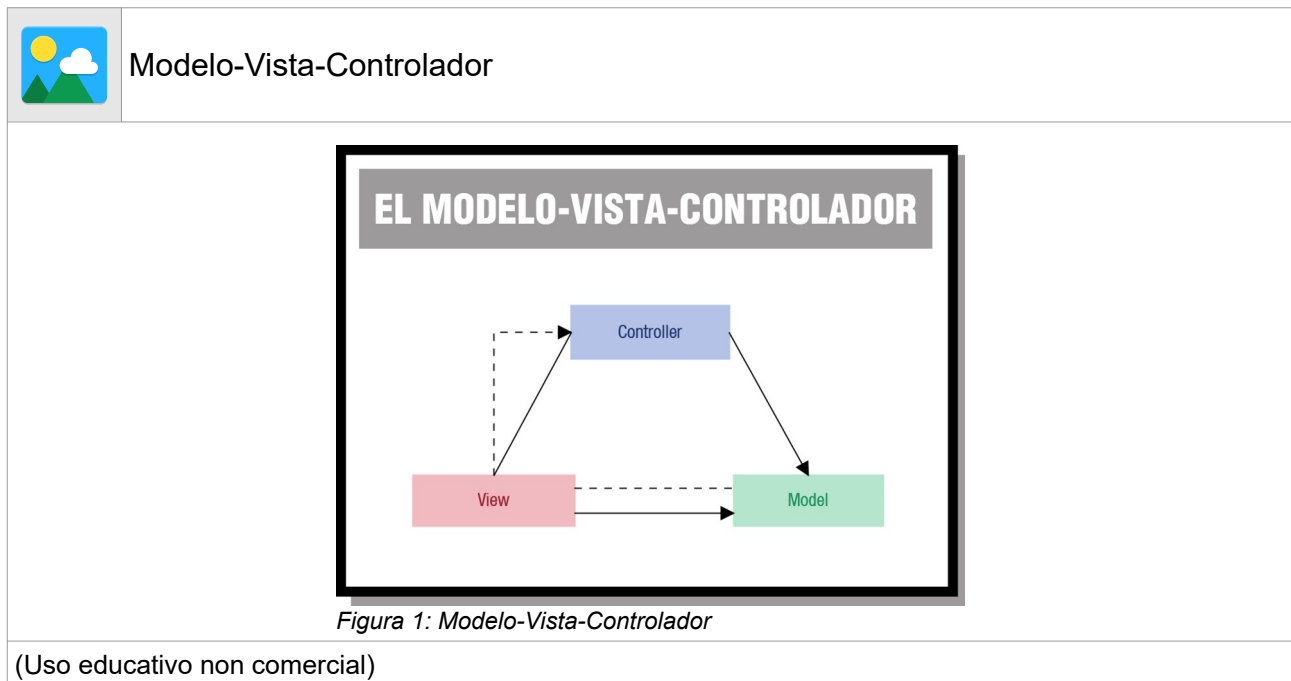
|                                                                                     |                                             |
|-------------------------------------------------------------------------------------|---------------------------------------------|
|    | Notas de introdución                        |
|    | A claración.                                |
|    | A rchivos de configuración, de rexistro ... |
|    | Casos de uso                                |
|    | Código fonte                                |
|    | A visosou advertencias                      |
|    | Capturas de pantalla, imaxes                |
|  | A ctividades.                               |
|  | Enlace recomendado                          |

|                                                                                     |                                                                                                                                                                                               |
|-------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | Iconas proporcionadas por Papyrus Development Team<br><a href="https://github.com/PapyrusDevelopmentTeam/papyrus-icon-theme">https://github.com/PapyrusDevelopmentTeam/papyrus-icon-theme</a> |
|-------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## 5.2 Técnicas e estándares. Modelo-Vista-Controlador

Utilizas habitualmente follas de cálculo, por exemplo Calc de OpenOffice.org? Nese caso, saberás que ao introducir datos nas celas, os mesmos datos podemos velos de varias formas. É posible ver os datos na folla de cálculo onde os introducimos, ou mediante un gráfico que pode ser de barras, circular etc.

Eses programas implementan o patrón Modelo-Vista-Controlador (MVC), o modelo constitúeno os datos que introducimos nas celas. As vistas encárganse de mostrar os datos da forma que se seleccione.



O MVC divide unha aplicación en tres compoñentes:

- Os datos da aplicación (**modelo**).
- A interface do usuario (**vista**).
- O **controlador**, o cal define a forma en que a interface reacciona á entrada do usuario.

Con isto conséguese separar os datos (modelo) da interface de usuario (vista), de maneira que os cambios na interface non afectan os datos e viceversa, é dicir, que os datos poden ser cambiados sen afectar á interface de usuario.

En ODOO, o MVC impleméntase da seguinte forma:

- O **modelo** son as táboas da base de datos.
- A **vista** son os arquivos XML que definen a interface de usuario do módulo.
- O **controlador** son os obxectos creados en Python.

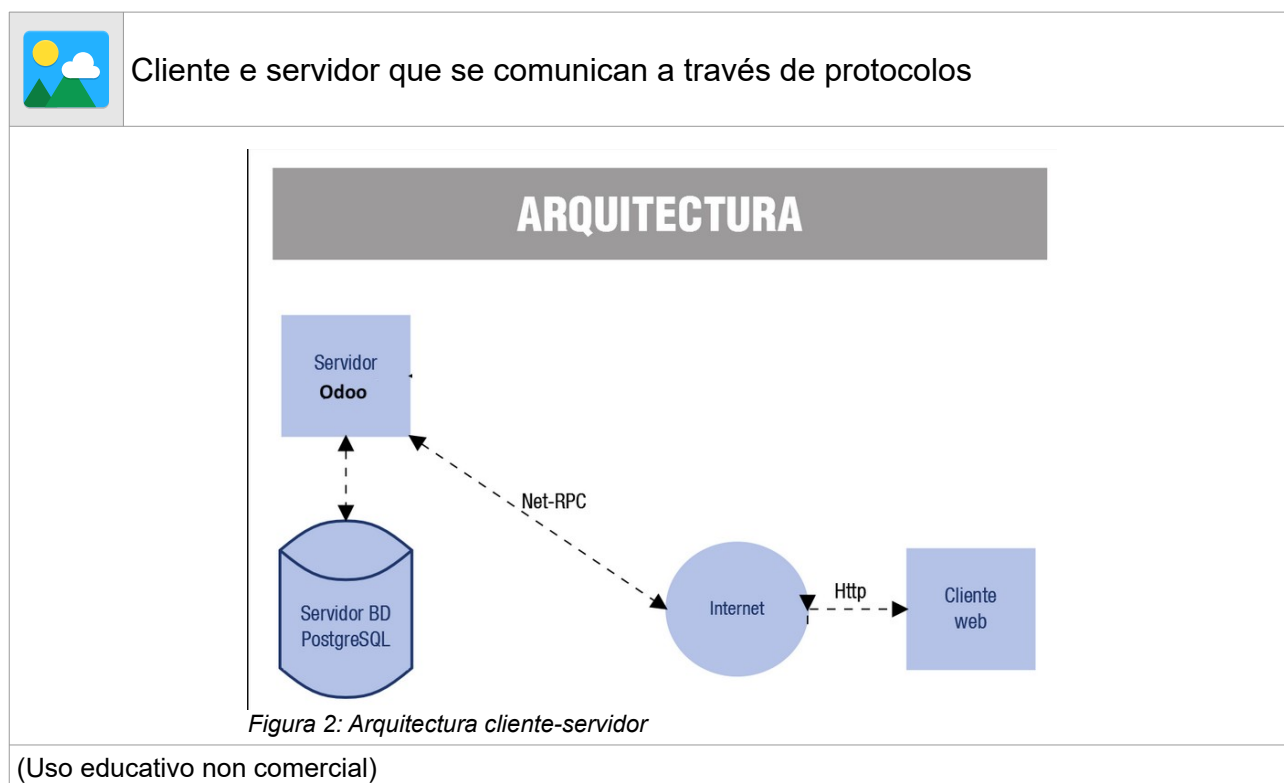
Por exemplo, no diagrama anterior, as frechas continuas significan que o controlador ten un acceso completo á vista e ao modelo, e as frechas descontinuas significan un acceso limitado. As razóns deste deseño son as seguintes:

Acceso de modelo a vista: o modelo envía unha notificación á vista cando os seus datos foron modificados, co fin de que a vista poida actualizar o seu contido. O modelo non necesita coñecer o funcionamento interno da vista para realizar esta operación. Con todo, a vista necesita acceder á partes internas do controlador.

Acceso de vista a controlador: a razón de que a vista teña limitado o acceso ao controlador é porque as dependencias que a vista ten sobre o controlador deben ser mínimas xa que o controlador pode ser substituído en calquera momento.

### 5.2.1 Especificacións técnicas para o desenvolvemento de compoñentes

Lembras na primeira unidade cando falabamos da arquitectura dos sistemas de planificación empresarial? Diciamos que a maioría están baseados nunha arquitectura cliente-servidor, leste é o caso de Odoo.



Odoo utiliza os protocolos XML-RPC ou Net-RPC para comunicación entre cliente e servidor. Basicamente o que fan é permitir ao cliente facer chamadas a procedementos remotos, ou sexa, executar código doutra máquina. No caso de XML-RPC, a función chamada, os seus argumentos e o resultado envíanse por HTTP e son codificadas usando XML. Net-RPC está dispoñible máis recentemente, está baseado en funcións en Python e é máis rápido.

ODOO funciona sobre un marco de traballo ou framework chamado OpenObject. Este framework permite o desenvolvemento rápido de aplicacións (RAD), sendo os seus principais elementos os seguintes:

- ORM: mapeo de bases de datos relacionais a obxectos Python.
- Arquitectura MVC (Modelo Vista Controlador).
- Deseñador de informes.
- Ferramentas de Business Intelligence e cubos multidimensionales.
- Cliente web.

### 5.2.2 Especificacións funcionais para o desenvolvemento de compoñentes

Como sabemos, Odoo componse dun núcleo e varios módulos dependendo das necesidades, por exemplo:

- **base**: é o módulo básico composto de obxectos como empresas (res.partner), direccións de empresas (res.partner.address), usuarios (res.user), moedas (res.currency) etc.
- **account**: xestión contable e financeira.
- **product**: produtos e tarifas.
- **purchase**: xestión de compras.
- **sae**: xestión de vendas.
- **mrp**: fabricación e planificación de recursos.
- **crm**: xestión das relacións con clientes e provedores.

Por cada módulo existe un cartafol co nome do módulo no directorio addons do servidor. Para crear un módulo teremos que crear un cartafol dentro dese directorio, e seguir os seguintes pasos:

1. Crear o arquivo de inicio de módulo: `__init__.py`
2. Crear o arquivo coa descrición do módulo: `__manifest__.py`.
3. Crear os arquivos Python coa definición de obxectos: `nome_modulo.py`, por medio de clases da linguaxe Python definimos o modelo e o controlador.
4. Vista ou vistas do obxecto `nome_modulo_nome_obxecto.xml`.

No cartafol **addons** ademais dos arquivos anteriores, pode haber outros cartafois como report, wizard, test, que conteñen arquivos do tipo de obxecto utilizado, neste caso, informes, asistentes e accesos directos, datos de proba etc.





El módulo basee constitúe o núcleo da aplicación, a partir do cal se instalan todos os demais módulos

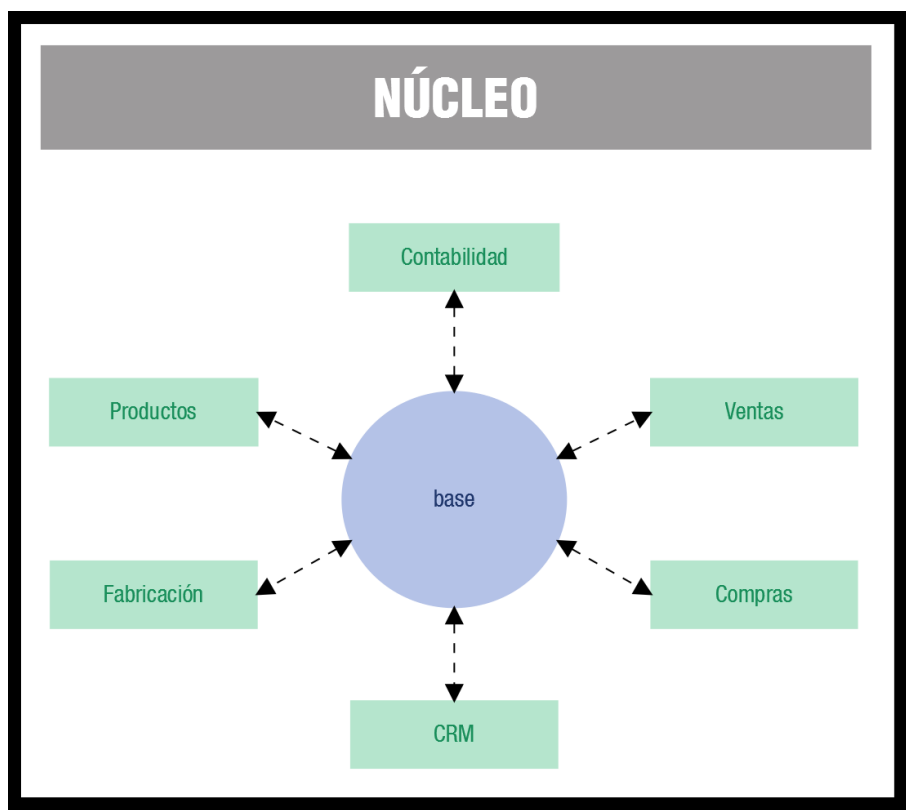


Figura 3: Módulo basee

(Uso educativo non comercial)

O arquivo `__manifest__.py` debe conter os seguintes valores dentro dun dicionario Python (estrutura de datos pechada con chaves `{ }`):

- **name:** nome do módulo.
- **version:** versión do módulo.
- **description:** unha descrición do módulo.
- **author:** persoa ou entidade que desenvolveu o módulo.
- **website:** sitio web do módulo.
- **license:** tipo de licenza do módulo (por defecto GPL).
- **depends:** lista de módulos dos que depende o módulo, o módulo basee é o máis usado xa que nel defínense algúns datos que son necesarios as vistas, informes etc.
- **init\_xml:** lista dos arquivos XML que se cargarán coa instalación do módulo.
- **installable:** determina se o módulo é instalable ou non.

## 5.3 Técnicas de optimización de consultas e acceso a grandes volumes de información

Un aspecto para ter en conta á hora de facer modificacións na aplicación e crear módulos que utilicen novos obxectos, é que estes obxectos baséanse en consultas á información existente na base de datos. Se desexamos mellorar os tempos de resposta do sistema, deberemos crear obxectos que utilicen consultas o máis optimizadas posibles. Podemos dicir que a optimización é o proceso de modificar un sistema para mellorar a súa eficiencia ou o uso dos recursos dispoñibles.

Cando manexamos grandes cantidades de datos, o resultado dunha consulta pode tomar un tempo considerable, obtendo non sempre unha resposta óptima. Dentro das técnicas de optimización de consultas podemos atopar as seguintes:

- **Deseño de táboas.** Á hora de crear novas táboas, asegurarnos de que non hai duplicidade de datos e que se aproveita ao máximo o almacenamento nas táboas.
- **Campos.** É recomendable axustar ao máximo o espazo nos campos para non desperdiciar espazo.
- **Índices.** Permiten procurar a unha velocidade notablemente superior, pero non debemos estar incitados a indexar todos os campos dunha táboa, xa que os índices ocupan máis espazo e tárdase máis ao actualizar os datos. Dous das razóns principais para utilizar un índice son:
  - É un campo utilizado como criterio de procuras.
  - É unha clave allea noutra táboa.
- **Optimizar sentenzas SQL.** Existen unha serie de regras para utilizar a linguaxe de consulta e modificación de datos que hai que contemplar. Estas regras refírense tanto á utilización das sentenzas de selección, como ás que realizan algunha inserción ou modificación na base de datos.
- **Optimizar a base de datos.** Tamén podemos conectarnos en modo comando á base de datos e utilizar sentenzas para optimizar os datos contidos na base de datos.

### 5.3.1 Operacións de consulta. Ferramentas

Para optimizar a base de datos necesitamos conectarnos con PostgreSQL en modo comando, os pasos serían os seguintes:

1. Cambiarnos ao usuario postgres. Isto debemos facelo porque temos que entrar no monitor interactivo cun usuario que exista en PostgreSQL:

```
$ sudo su postgres
```

2. Entramos no monitor interactivo de PostgreSQL chamado psql:

```
$ psql
```

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

3. Unha vez dentro do monitor interactivo, o Prompt `postgres=#` significa que o monitor está listo e que podemos escribir comandos.
4. Saír do monitor de PostgreSQL, co comando `\q`.

Entre os comandos que podemos utilizar temos:

- `\h` ----> Axuda.
- `\l` ----> Mostra as bases de datos existentes.
- `\c [nome_bd]` ----> Conectámonos coa base de datos que queiramos.
- `\d` ----> Mostra as táboas existentes na base de datos.
- `\d [nome_táboa]` ----> Mostra a descrición dunha táboa, ou sexa, os campos que contén, de que tipo son etc.
- `VACUUM VERBOSE ANALYZE [táboa]` ; ----> Limpa e analiza bases de datos.
- `\q` ----> Saímos do editor de consultas.

### 5.3.2 Sistemas batch inputs. Xeración de programas de extracción e procesamento de datos.

O termo batch-input procede da utilización en sistemas SAP dun método utilizado para transferir grandes cantidades de datos a un sistema ERP. Existen dúas formas de facer un batch-input:

- **Método clásico.** Método asíncrono, é dicir, que se procesan os datos pero actualízanse máis tarde. Ten a característica de que xera un arquivo de mensaxes ou log para tratarse erros a posteriori.
- **Método "call transaction".** Método on-line usado para dar de alta rapidamente poucos rexistros. É un método síncrono, non xera log é moito máis rápido pero pouco útil para gran cantidade de datos.

Un proceso batch-input componse de dúas fases:

- **Fase de xeración.** É a fase en que se xera o arquivo batch-input cos datos para introducir ou modificar.

- **Fase de procesamento.** O arquivo batch-input execútase, facéndose efectivas as modificacións na base de datos.

## 5.4 Linguaxe proporcionada polos sistemas ERP-CRM

A maioría de aplicacións de planificación empresarial están construídas con linguaxes de programación modernos e, en moitos casos, orientados a obxectos. Existen aplicacións ERP que proporcionan linguaxes propietarias, que non son linguaxes estándar e requiren que o programador se forme nesa linguaxe só e exclusivamente para manexar o ERP.

A linguaxe de programación utilizada en Odoo é a linguaxe Python.

Esta linguaxe foi creado por Guido van Rossum a principios dos anos 90 no Centro para as Matemáticas e a Informática (CWI, Centrum Wiskunde & Informatica), nos Países Baixos. Guido segue sendo o autor principal de Python.

En 2001, creouse a Python Software Foundation (PSF), unha organización sen fins de lucro creada especificamente para posuír a propiedade intelectual sobre a licenza e promover o uso da linguaxe. Como membros patrocinadores desta organización están Google, Canonical e Microsoft.

A licenza de Python é de código aberto (Python Software Foundation License, PSFL), compatible con GPL. É unha linguaxe moi sinxela de utilizar cunha sintaxe moi próxima á linguaxe natural, polo que se considera un das mellores linguaxes para empezar a programar.

Python é utilizado por Google, Yahoo, a NASA ou por exemplo en todas as distribucións Linux, onde os programas escritos nesta linguaxe representan unha porcentaxe cada vez maior. Aínda que ver en profundidade esta linguaxe levaríanos máis tempo do esperado, imos ver algunhas das súas características máis importantes para poder crear os nosos propios módulos en Odoo.

### 5.4.1 Características e sintaxes da linguaxe

As principais características de Python son:

- **Sintaxe moi sinxela**, o cal facilita unha rápida aprendizaxe da linguaxe.
- **Linguaxe interpretada.** Os programas en Python execútanse utilizando un programa intermedio chamado intérprete. O código fonte tradúcese a un arquivo chamado bytecode con extensión .pyc ou .pyo, que é o que se executará en sucesivas ocasións.
- **Tipado dinámico.** Esta característica quere dicir que os datos non se declaran antes de utilizalos, senón que o tipo dunha variable determínase en tempo de execución, segundo o dato que se lle asigne.
- **Fortemente tipado.** Cando unha variable é dun tipo concreto, non se pode usar coma se fóra doutro tipo, a non ser que se faga unha conversión de tipo. Se a variable é numérica enteira, por exemplo, non poderei asignarlle un valor real.

Noutras linguaxes, o tipo da variable cambiaría para adaptarse ao comportamento esperado, aínda que iso pode dar máis lugar a erros.

- **Multiplataforma.** O intérprete de Python está dispoñible para unha gran variedade de plataformas: Linux, Windows, Macintosh etc.
- **Orientado a obxectos.** Os programas están formados por clases e obxectos, que son representacións do problema no mundo real. Os datos, xunto coas funcións que os manipulan, son parte interna dos obxectos e non están accesibles ao resto dos obxectos. Por tanto, os cambios nos datos dun obxecto só afectan as funcións definidas para ese obxecto, pero non ao resto da aplicación.



Guido van Rossum, creador de Python, na conferencia OSCON 2006



Figura 4: Guido van Rossum

Docsearls (CC BY-SA)

Para facer o primeiro programa en Python non necesitamos coñecer moito sobre a linguaxe, por exemplo, podemos facer o famoso "**Ola mundo**" co que se adoita comezar a estudar case todas as linguaxes, simplemente coa seguinte liña:




Ola mundo

```
print ('Hello World!')
```

Se escribimos isto no intérprete de Python executarase e imprimírase por pantalla a mensaxe "Ola mundo". Xa temos o noso primeiro programa!. Probémolo. O intérprete de Python está instalado por defecto na maioría das distribucións de Linux, para saber se o temos instalado simplemente temos que introducir Python nun terminal, e apareceranos unha mensaxe coa versión que temos instalada e o prompt >>>, que indica que está a

esperar código do usuario. Podemos saír escribindo `exit()`, ou coa combinación de teclas **Control + D**.


 Terminal de ArcoLinux executando Ola Mundo en Python

```
fermigo@NUC13ANH7:~  
  
Archivo  Editar  Ver  Buscar  Terminal  Ayuda  
[fermigo@NUC13ANH7 ~]$ python3  
Python 3.11.6 (main, Nov 14 2023, 09:36:21) [GCC 13.2.1 20230801] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> print ('Hello World!')  
Hello World!  
>>> exit()  
[fermigo@NUC13ANH7 ~]$
```

Figura 5: Terminal de ArcoLinux executando Ola Mundo en Python

Terminal de ArcoLinux.

Tamén podemos gardar o código nun arquivo chamado **programa.py**, e executalo coa orde **python3 programa.py**.

 Terminal de ArcoLinux executando un programa Python

```
fermigo@NUC13ANH7:~  
  
Archivo  Editar  Ver  Buscar  Terminal  Ayuda  
[fermigo@NUC13ANH7 ~]$ cat hello.py  
print ('Hello World!')  
[fermigo@NUC13ANH7 ~]$ python3 hello.py  
Hello World!  
[fermigo@NUC13ANH7 ~]$
```

Figura 6: Terminal de ArcoLinux executando un programa Python

Terminal de ArcoLinux.

Os comentarios na linguaxe escríbense co carácter **#**, as asignacións de valores ás variables co carácter igual (**=**). Para crear unha función utilizamos o seguinte código, hai que ter en conta que a sangría do texto serve para delimitar que instrucións van dentro da función:



## Función en python

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

É dicir, nesta linguaxe non hai delimitador de sentenza como o punto e como de Java ou C, nin delimitador de bloques, como as chaves destes mesmas linguaxes.

### 5.4.2 Declaración de datos. Tipos básicos

Como xa comentamos, en Python non se declaran as variables como tal, senón que se utilizan directamente. Podemos declarar variables dos seguintes tipos básicos:

- **Números enteiros.** Os números enteiros son o cero e aqueles números positivos ou negativos que non teñen decimais. En Python pódense representar mediante o tipo `int` que utiliza 32 ou 64 bits, dependendo do tipo de procesador, ou o tipo `long`, que permite almacenar números de calquera precisión, estando limitados só pola memoria dispoñible na máquina.
  - `# declaración dun número enteiro`
  - `a = 2`
  - `# declaración dun número enteiro longo`
  - `a = 2L`
- **Números reais.** Son os que teñen cómaa decimal. Exprésanse mediante o tipo `float`.
  - `b = 2.0`
- **Números complexos.** Son un tipo de dato especial composto por unha banda real e unha parte imaxinaria.
  - `complexo = 2.0 + 7.5 j`
- **Cadeas.** É texto encerrado entre comiñas simples ou dobres.
  - `mensaxe = "Bienvenido a Python"`
- **Booleanos.** Só poden ter dous valores `True` ou `False`.

De entre os operadores máis importantes para traballar con estes datos podemos destacar os seguintes:

- **Operadores aritméticos:** suma (+), resta (-), multiplicación (\*), división(/), división enteira (//), expoñente (\*\*), módulo (%).

- **Operadores booleanos:** and, or, not.
- **Operadores relacionais:** igual (==), distinto (!=), menor (<), maior (>), menor igual(<=), maior ou igual (>=).

### 5.4.3 Estruturas de programación. Coleccións

Python ten unha serie de estruturas de datos que se utilizan amplamente. Son tipos avanzados de datos, que reciben o nome de Coleccións, a saber:

- **Listas.** Son coleccións ordenadas de datos, noutras linguaxes coñécense como arrays ou vectores. Poden ter calquera tipo de dato.

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

Accédese a un elemento concreto da lista usando o índice da súa posición (empezando por cero) entre corchetes.

```
a = l [0]          # a vale 3
```

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

- **Tuplas.** Son parecidas ás listas, coa diferenza que son máis lixeiras polo que se o uso é básico utilízanse mellor tuplas. Os elementos van entre parénteses en lugar de entre corchetes.

```
t = ( "ola", 2, False)
```

```
b = t[1]          # b vale 2
```

- **Dicionarios.** Son coleccións que relacionan unha clave e un valor. Para acceder a un valor utilízase o operador [ ], igual que para as filas e tuplas, e dentro del a clave á que queremos acceder.

```
d = {"Nome": "Alberto",
```

```
"Apelido" : "Contador",
```

```
"Vitorias" : [ 2007 , 2008 , 2009 ] }
```

```
x = d ["Nome"]          # x vale "Alberto"
```

```
x = d["Apelido"]        # x vale "Contador"
```

```
x = d["Vitorias"][1]    # x vale 2008
```

As listas son obxectos mutables, é dicir, podemos modificar cada un dos seus compoñentes. As tuplas con todo son non mutables, é dicir, que non podemos modificalas unha vez que se crearon, por exemplo, se tentamos facer isto:

```
tupla = ( 1 , 'ola', 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
```

O intérprete de Python dános un erro porque as tuplas non se poden modificar.

#### 5.4.4 Sentenzas da linguaxe

A sintaxe das estruturas de programación en Python é a seguinte:

- **Estrutura condicional.** A sintaxe máis común da estrutura condicional é un `if` seguido da condición para avaliar, seguida de dous puntos, e a seguinte liña sangrada, para indicar o código que queremos que se execute se se cumpre a condición. No caso de que non se cumpra a condición, executamos a sentenza detrás do `else`:

```
if (numero / 2 == 0):
    print ("O numero é par")
else:
    print ("O numero é impar")
```

- **Bucles.** Permítenos executar un fragmento de código un número de veces, ou mentres se cumpra unha determinada condición.

```
while)
```

O bucle `while` (mentres) repite todo o seu bloque de código mentres a expresión avaliada sexa certa. Detrás da condición hai que poñer dous puntos, e debaixo as sentenzas que van dentro do `while`.

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

O funcionamento é sinxelo, o programa avalía a condición, e se é certa, executa o código que hai dentro do `while`, cando acaba volve avaliar a condición e se é certa executa novamente o código, e así ata que a condición devolva `false`, nese caso, o programa continuaría executando as instrucións seguintes ao bucle. `for ... in`

A sentenza `for` utilízase para percorrer secuencias de elementos. Se utilizamos un tipo secuencia, como é a lista no seguinte exemplo:

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

Vemos que o resultado é:

```
Maria
Pepe
Juan
```

O que fai o bucle é que para cada elemento que haxa na secuencia, executa as liñas de código que teña dentro, neste caso a sentenza `print`. O que fai a cabeceira do bucle é obter o seguinte elemento da secuencia milista e almacenalo nunha variable de nome `elemento`. Por esta razón na primeira iteración do bucle elemento valerá "Maria", na segunda "Pepe", e na terceira "Juan".

### 5.4.5 Chamadas a funcións

Unha función é un conxunto de instrucións ou programa que realiza unha tarefa específica e que devolve un valor. As funcións axúdannos a reutilizar código, e a dividir o programa en partes máis pequenas con obxecto de facelo máis organizado, lexible e máis fácil de depurar.

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

Co código anterior estamos a declarar a función `cadrado` que, como o seu nome indica, calcula o cadrado dun número. O número que aparece entre paréntese é o valor que lle pasamos á función para que poida calcular o seu cadrado, e recibe o nome de parámetro. As funcións poden ter un, varios ou ningún parámetro.

Cando executamos a función dise que estamos a facer unha chamada á función, e facémolo utilizando o seu nome e os parámetros entre parénteses:

```
cadrado (4)
```

O resultado de chamar á función será que se imprimirá por pantalla o valor 16.

As funcións poden ou non devolver un valor como resultado da súa execución. Podemos transformar o exemplo anterior para que devolva o resultado en lugar de mostralo por pantalla. O código sería o seguinte:

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

E chamariamos á función coa seguinte instrución:

```
print (cadrado(4))
```

O resultado sería o mesmo, só que agora a función devolve o valor, e ese valor é o que mostra por pantalla a instrución `print`.

## 5.4.6 Clases e obxectos

Ata o de agora vimos a sintaxe básica de Python. Pero para entender ben como está formado un módulo en Odoo, necesitamos saber tamén que son as **clases** e os **obxectos**. Se programaches antes cunha linguaxe orientada a obxectos, non terás ningún problema en entender a definición de ambos os conceptos.

Un programa orientado a obxectos está formado por clases e obxectos. Cando temos un problema que resolver, dividímolos nunha serie de obxectos e son eses obxectos os que describimos no noso programa. O programa consiste nunha serie de interaccións entre eses obxectos. As clases son persoais a partir das cales se crean os obxectos, cando creamos un obxecto a partir dunha clase di-se que ese obxecto é unha **instancia** da clase.

As clases teñen unha serie de atributos ou campos que as definen, así como unha serie de métodos para facer operacións sobre eses campos. Cando creamos un obxecto dámoslle valor a eses atributos. En Python as clases defínense mediante a palabra clave `class` seguida do nome da clase, dous puntos (`:`) e a continuación, sangrado, o corpo da clase.

Por exemplo, imos crear unha clase `Persoa` que ten como atributos o **nome** e a **idade**:

```
class persoa:
    def __init__(self, nome, idade):
        self.nome = nome
        self.idade = idade
    def mayordeedad(self):
        if self.idade > 18:
            print ("É maior de idade")
        else:
            print ("Non é maior de idade" )
```

Na definición de clase anterior vemos que estamos a definir un método `__init__`. O nome deste método non é casual, senón que se utiliza para inicializar o obxecto. Cando instanciamos un obxecto a partir da clase, será o primeiro método que se executará. Tamén vemos que os dous métodos da clase teñen como primeiro argumento `self`. Este argumento serve para referirnos ao obxecto actual, e utilízase para poder acceder aos atributos e métodos do obxecto.

Para crear un obxecto da clase utilizamos a seguinte instrución:

```
empregado = persoa("Javier",32)
```

Agora que xa creamos o noso obxecto, podemos acceder aos seus atributos e métodos mediante a sintaxe **obxecto.atributo** e **obxecto.método**:

```
print (empregado.nome, "ten", empregado.idade)
```

### 5.4.7 Módulos e paquetes

Cando saímos do intérprete e volvemos entrar empezamos desde o principio, se creamos algunha variable ou programa este desaparece e temos que volver escribir o código. Para conservar os programas que facemos, podemos crear arquivos con extensión `.py`, de maneira que sempre teñamos o código dispoñible para executalo co intérprete e que non sexa necesario volver introduci-lo.

Por outra banda, os programas ás veces son moi grandes e ao dividilos en partes fanse máis fáciles de manter. Agrúpase o código relacionado e se garda en arquivos diferentes. Cada arquivo é un módulo en Python. Noutras palabras, un módulo é un arquivo que contén definicións e declaracións de Python.

As razóns para utilizar módulos son principalmente as seguintes:

- Facer o código máis fácil de manter.
- Reutilización de código, por exemplo, unha función que queiramos utilizar en varios programas.

Se queremos utilizar un módulo noutro arquivo, temos que importalo coa seguinte instrución:

```
import nome_modulo
```

**Un paquete é unha colección de módulos relacionados.** Mentres os módulos axudan a organizar o código, os paquetes axudan a organizar os módulos. A nivel físico, os módulos son os arquivos, e os paquetes son os directorios.

Para facer que Python trate a un directorio como un paquete é necesario crear un arquivo `__init__.py` no devandita cartafol. Para facer que un módulo pertenza a un paquete hai que copiar o arquivo `nome_modulo.py` que define o módulo no directorio do paquete. Para importar paquetes utilízase a sentenza `import` igual que cos módulos.

### 5.4.8 Librerías de funcións (APIs)

Unha API, ou Biblioteca de Clases, é un conxunto de clases útiles que teñen a disposición os programadores para utilizar nos seus programas. Python proporciona unha API estándar, cuxo código fonte está a libre disposición para as principais plataformas desde o sitio web de Python. Tamén é posible acceder a moitos módulos libres de terceiros na mesma páxina web, así como a programas, ferramentas e documentación adicional.

Na ligazón expresada máis abaixo atoparás unha referencia a toda a Biblioteca estándar de módulos de Python, entre os que destacan:

- `os` :prové de diferentes funcións do sistema operativo, como creación de arquivos e directorios, ler ou escribir dun arquivo, manipular roteiros etc.
- `sys`: permite acceder a información sobre o intérprete de Python, por exemplo o prompt do sistema, datos de licenza e en xeral calquera parámetro ou variable que teña que ver co intérprete.

- `datetime`: módulos para a manipulación de datas e horas.
- `math`: funcións matemáticas.

### 5.4.9 Inserción, modificación e eliminación de datos en obxectos

Neste apartado imos ver como están estruturados os módulos e como crear un primeiro módulo sinxelo que cre un obxecto na aplicación para inserir, modificar ou eliminar datos na base de datos.

Os módulos de Odoo gárdanse dentro do cartafol **addons**, que para nosa versión de Ubuntu atópase en **/usr/lib/python3/dist-packages/odoo/addons**. Se instalaches a aplicación desde un arquivo da internet en lugar de desde Synaptic, ou a túa versión da aplicación ou Ubuntu é diferente, poida que ela roteiro do directorio **addons** sexa diferente.

O que temos que facer é crear un cartafol para o noso módulo dentro do directorio **addons**. Os arquivos que ten que conter o cartafol son:

- **\_\_init\_\_.py**. Necesario para que o cartafol trátase como un paquete en Python, contén os import de cada arquivo do módulo que conteña código Python, neste caso, o arquivo `nome_modulo.py`.
- **\_\_manifest\_\_.py**. Contén un diccionario Python coa descrición do módulo, como quen é o autor, a versión do módulo ou cales son os outros módulos dos que depende.
- **nome\_modulo.py**. Neste arquivo creamos a clase definindo os campos que vai ter. Ao crear a clase estamos a crear o modelo (unha táboa na base de datos) e tamén estamos a crear o controlador, porque cando creamos unha clase estamos a definir o comportamento que ten.
- **nome\_modulo\_view.xml**. Neste arquivo definimos a vista do noso módulo, ou do obxecto que vai crear o noso módulo. Para crear este arquivo necesitamos ter certos coñecementos de XML, ou ben coller unha vista doutro obxecto e a partir dela crear a do novo obxecto.

Tamén existe outra forma de crear módulos sen necesidade de ningún desenvolvemento, que é utilizando o módulo **basee\_module\_record**. Se instalamos este módulo, teremos a opción de gravar todas as accións que realicemos na aplicación. Se utilizaches algunha vez as macros nun procesador de textos ou nunha folla de cálculo saberás do que estamos a falar. O módulo `basee_module_record` funciona de forma parecida. Por exemplo, podemos crear un novo obxecto e asignarlle un menú, todo iso gravaríase nun arquivo comprimido. Realmente ese arquivo comprimido o que vai conter é un módulo, con todos os arquivos indicados anteriormente. Se con posterioridade collemos eses arquivos e copiámoslos ao cartafol **addons**, poderemos instalalos na aplicación como calquera outro módulo. Ao instalalo, o resultado sería que aparecería ese novo obxecto e menú na nosa aplicación.

## 5.5 Contornas de desenvolvemento e ferramentas de desenvolvemento en sistemas ERP-CRM

A ferramenta máis importante cando estamos a programar sempre é unha boa contorna de desenvolvemento. Se queremos unha resposta rápida e o programa é pequeno podemos utilizar o intérprete, como viñemos facendo ata o de agora. Con todo, cando o programa vai sendo máis grande, é moito máis cómodo utilizar unha contorna de desenvolvemento desde a cal podemos facer todas as operacións de maneira gráfica.

Unha contorna de desenvolvemento é un programa que inclúe todo o necesario para programar nun ou varias linguaxes. Ademais dun editor de textos especializado como ferramenta central, inclúen navegadores de arquivos, asistentes de compilación, depuradores etc.

Entre as contornas de desenvolvemento para traballar con Python atopámonos cun dos máis sinxelos, ÍDELLE, dispoñible na maioría das plataformas. En Ubuntu pódese descargar desde Synaptic. Esta contorna basicamente o que permite é revisar a sintaxe do noso módulo e executalo. Tamén inclúe opcións de depuración.

Cando o executamos aparécenos unha xanela cun sinxelo menú. Desde el poderemos abrir o arquivo con código python, e aparecerá unha nova xanela onde prodremos revisar a sintaxe desde o menú Run/Check module ou executar o programa desde o menú Run/Run module, entre outras opcións.

Noutra orde de cousas, temos a extensión ou plugin de Odoo para o editor Gedit. Esta extensión permite utilizar o editor de texto como unha contorna de desenvolvemento para Odoo. A principal vantaxe é que engade fragmentos de código típico, o cal facilita en gran medida non ter que lembrar a sintaxe.

Finalmente, dentro das contornas de desenvolvemento máis completos para traballar con Python temos Eclipse. Eclipse ademais incorpora unha serie de persoais con código, o cal facilita moito a creación dos módulos en Odoo.

### 5.5.1 Depuración dun programa

As ferramentas de depuración cando estamos a desenvolver un programa axudan a detectar erros no código, mostrando información de que tipo de erro é ou porqué prodúcese. As ferramentas de depuración adoitan formar parte das contornas de desenvolvemento.

A linguaxe Python incorpora un depurador dentro da súa Biblioteca de módulos estándar chamado pdb. Soporta puntos de ruptura e execución paso a paso do código, inspeccionar valores de variables e outras opcións de depuración. Con todo, como en todas as linguaxes, se utilizamos unha ferramenta gráfica para o proceso de depuración, seguramente aforraremos moito tempo.

Dentro da contorna de desenvolvemento ÍDELLE poderemos depurar os nosos programas. Esta contorna está dispoñible para calquera distribución e sistema operativo. Para activar o módulo de depuración facemos clic no menú Debug/Debugger. Unha vez que estea o

modo de depuración activado o funcionamento é como o de calquera depurador. Co botón dereito establecemos unha parada na sentenza do código que queremos avaliar, de maneira que ao executar o programa o depurador pararase nesa instrución, e poderemos analizar o valor das variables nese punto.

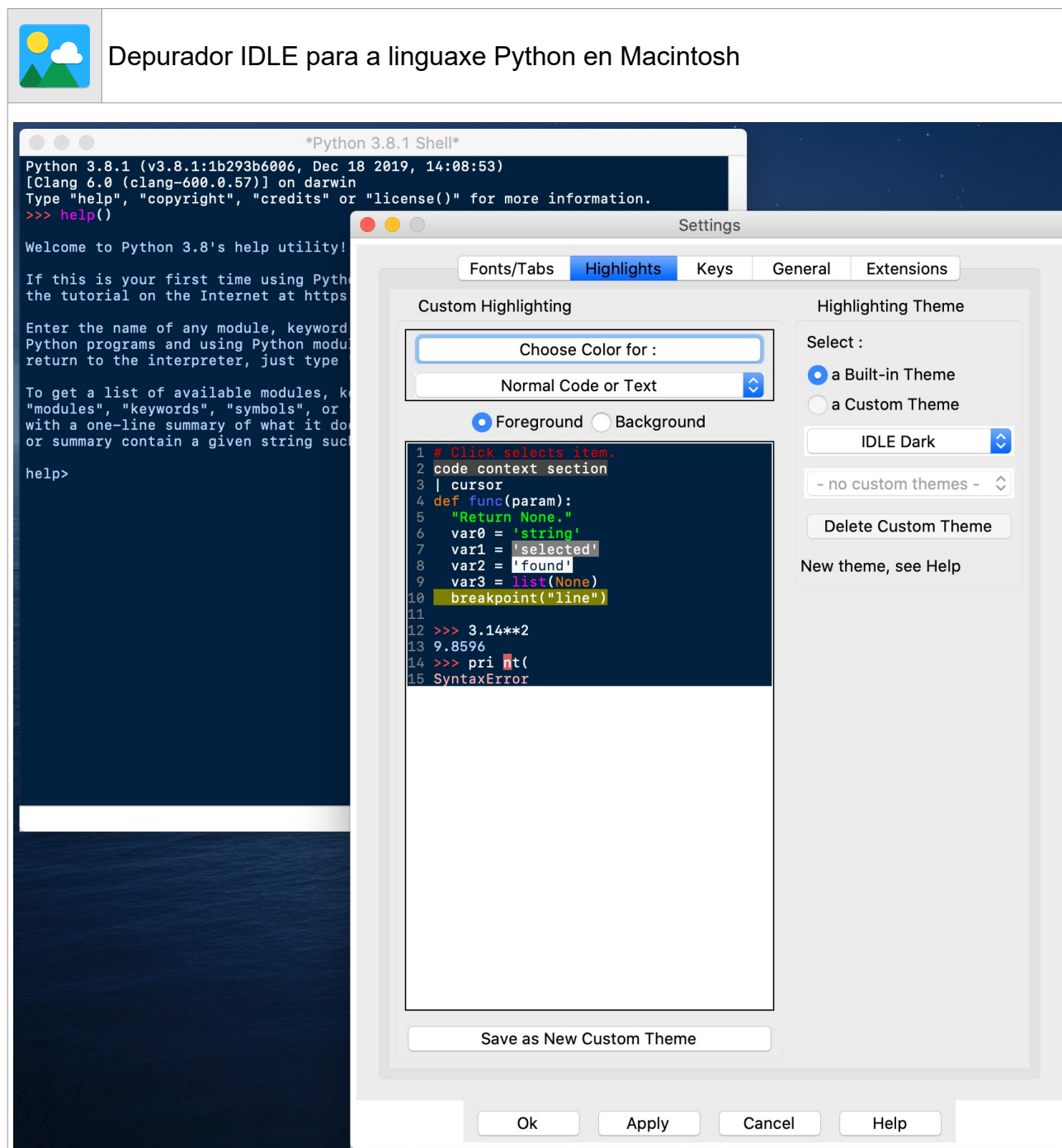


Figura 7: Depurador IDLE para a linguaxe Python en Macintosh

Maćko (CC BY-SA)

As accións que podemos realizar co depurador son as seguintes:

- **Go**: fai que a execución do programa continúe ata que atope un punto de ruptura.
- **Step**: vai executando o programa liña a liña.



- **Over:** fai que a sentenza actual sexa completamente executada sen parar en ningunha función aniñada.
- **Out:** fai que se compute desde o punto actual ata o fin da función actual e que esta se termine.
- **Quit:** finaliza a execución do programa.

### 5.5.2 Manexo de erros

Cando o noso programa falla, xérase un erro en forma de excepción. As excepcións son erros detectados polo intérprete durante a execución do programa. Por exemplo, podemos tentar dividir por cero ou acceder a un arquivo que non existe, entón o programa xera unha excepción avisando ao usuario de que se produciu un erro. O noso labor como programadores é escribir o código de maneira que se contemplen esas posibles excepcións, e actúese en consecuencia. Se o noso programa non contempla o manexo de excepcións, cando ocorra algunha o que pasará é que se xerará o erro e o programa deixará de executarse ou non terá o funcionamento esperado.

En Python utilízase unha construción **try-except** para capturar e tratar as excepcións. Dentro do bloque **try** sitúase o código que cremos que podería producir unha excepción, e dentro do bloque **except**, escribimos o código que se executará se se produce dita excepción.

Vexamos un pequeno programa que lanzaría unha excepción ao tentar facer unha división entre 0:

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

dividir(3,0)
```

Se o executamos no intérprete obtemos a seguinte saída de erro:

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

O que nos di a mensaxe é que é unha excepción, **ZeroDivisionError**, e a descrición do erro: **"integer division or modulo by zero"** (módulo ou división enteira entre cero).

Se escribimos o mesmo programa, pero teniendo en cuenta o manexo de excepcións, o código quedaría como segue:

```
try:
    def dividir(a, b):
        return a / b

    dividir(1,0)
except:
```



```
print ("Ocorreu un erro")
```

## 5.6 Formularios e informes nun sistemas ERP-CRM

Os formularios constitúen a interface dun módulo. Unha vez creado o obxecto do módulo, ou sexa, o arquivo `nome_modulo.py`, o que debemos facer é crear os menús, accións, vistas (formulario ou outro tipo) para poder interactuar co obxecto. Estes elementos descríbense no arquivo `nome_modulo_view.xml`.

Dicir que é importante lembrar que o nome do ficheiro XML atópase listaxe no atributo `update_xml` do ficheiro `__terp__.p`, `__openerp__.py` a partir da versión 5.0, do módulo.

Os informes poden ser estatísticos, para os que temos o módulo `basee_report_creator` que permite crear listaxes, informes e gráficos por pantalla, e poden ser informes impresos, nese caso existen diversas ferramentas para xeralos.

### 5.6.1 Arquitectura de formularios e informes. Elementos

Os formularios e demais vistas da aplicación son construídos de forma dinámica pola descrición XML da pantalla do cliente. Podemos dicir que o linguaxe XML é unha metalinguaxe que utiliza etiquetas do tipo `etiqueta_contido_fin etiqueta` para expresar información estruturada.

Unha etiqueta consiste nunha marca feita no documento, que fai referencia a un elemento ou pedazo de información. As etiquetas teñen a forma `<nome>`, onde `nome` é o nome do elemento que se está sinalando.

No caso de Odoo, a estrutura do arquivo `nome_modulo_view.xml` e o ficheiro de definición `__manifest__.py` é a seguinte:

```
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 etiqueta 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="ou_view_nocontent_smiling_face">Create the first course
        </p>
    < /field>
</record>

<!-- top level menu: non 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 ide location:
    action="openacademy.course_list_action"
    It is not required when it is the same module -->

</odoo>

```

Cada tipo de rexistro, fai referencia a un obxecto diferente. Os rexistros defínense coa etiqueta `<record>` `</record>`, que indican o inicio e fin da descrición do rexistro. Dentro van os campos `field` que se utilizan para definir as características do rexistro. Seguindo a estrutura anterior, un exemplo de creación de vista de tipo formulario sería o seguinte:

```

<record model="ir.ui.view" id="view_axenda_form">
<field name="name">axenda</field>
<field name="model">axenda</field>

```

```

<field name="type">form</field>
<field name="priority" eval="5"/>
<field name="arch" type="xml">
<form string="Axenda">
<field name="nome" select="1"/>
<field name="telefono" select="1" />
</form>
</field>
</record>

```

O campo name fai referencia ao nome da vista e o campo name="model" fai referencia ao obxecto do modelo, é dicir, a táboa na base de datos, do cal vai mostrar a información. Vemos que a vista está formada por dous campos: **nome** e **telefono**.

De igual forma podemos definir o **elemento de menú** para acceder ao obxecto axenda:

```

<menuitem name="Axenda" ide="menu_axenda_axenda_form"/>
<menuitem name="Axenda" ide="menu_axenda_form" parent="axenda.menu_axenda_axenda_form"
action="action_axenda_form"/>

```

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

```

<record model="ir.actions.act_window" ide="action_axenda_form">
<field name="res_model">axenda</field>
<field name="domain">[]</field>
</record>

```

## 5.6.2 Ferramentas para a creación de formularios e informes

Existen diferentes ferramentas para a creación de formularios e informes en Odoo. Para empezar hai que distinguir entre a creación de formularios e a creación de informes. A creación de formularios como xa vimos realízase en arquivos XML co nome nome\_modulo\_view.xml. A creación de informes realízase con outras ferramentas. Xa vimos como crear informes co módulo basee\_report\_creator e coa extensión de OpenOffice.org para Opdoo. Existe outra forma de crear informes, e é mediante a utilización da librería Jasper Reports.

Jasper Reports é unha librería para a xeración de informes. O seu funcionamento consiste en xerar un arquivo XML coa descrición do informe para crear, que é tratado para xerar un documento nun formato de saída, por exemplo PDF ou HTML.