

SISTEMAS DE GESTIÓN EMPRESARIAL

UNIDAD 5: Desarrollo de módulos para un sistema ERP-CRM



ÍNDICE

5.1. Introducción

5.2. El lenguaje de programación Python.


5.3. Desarrollo de módulos para OpenERP.


5.1. Introducción.


- En esta unidad vamos a abordar la creación de módulos propios para **OpenERP**.
- Dividiremos el capítulo en dos partes bien diferenciadas, la primera centraremos en el lenguaje **Python** en la versión 2.7. En la segunda parte crearemos módulos propios integrándolos dentro de un sistema **OpenERP** funcional.
- La razón para utilizar **Python** es que **OpenERP** está programado en **Python**.
- Utilizaremos herramientas adicionales como **Eclipse** y **Dia** para construir los módulos.

5.2. El lenguaje de programación Python.

- **Python** es un lenguaje de programación creado por Guido van Rossum a principios de los 90 cuyo nombre está inspirado en el grupo de cómicos ingleses “Monty Python”.
- Es un lenguaje **interpretado multiplataforma** cuyo código fuente se convierte en código objeto (ficheros con extensiones **.pyc** o **.pyo**) la primera vez que se ejecuta, lo que le permite ser flexible como los lenguajes de script y veloz como un lenguaje compilado.
- Permite no declarar las variables para usarlas pero es obligatorio inicializarlas a algún valor antes de su utilización (esta característica se conoce como **tipado dinámico**).

- 
- No se permite tratar a una variable como si fuera de un tipo distinto al que tiene, es necesario convertir de forma explícita dicha variable al nuevo tipo previamente.
 - El intérprete de **Python** está disponible en multitud de plataformas (UNIX, Solaris, Linux, DOS, Windows, OS/2, Mac OS, etc.) por lo que si no utilizamos librerías específicas de cada plataforma nuestro programa podrá correr en todos estos sistemas sin grandes cambios.
 - El lenguaje **es orientado a objetos** definiendo **todo** como un **objeto** (incluso las funciones son objetos que incorporan atributos). Permite la creación de **clases** y **objetos**, implementa la **herencia** pero no limita el acceso a las **propiedades** o **métodos** que definamos.

- 
- En **Python** no existen mecanismos de definición de bloques explícitos, se crean con el sangrado obligatorio del código.
 - Un bloque comienza con el carácter dos puntos (:) y todo lo que debe ejecutarse dentro deberá de estar sangrado en el código con una serie de espacios en blanco.
 - Toma mucha importancia el número de espacios en blanco que usemos ya que deberán de tener los mismos todas las líneas del bloque si no se generará un error.
 - Un fallo muy común es el **confundir el tabulador con el espacio**, el lenguaje lo interpreta de forma diferente.

- 
- Existen **diferentes implementaciones de Python** según el lenguaje base que se haya utilizado. Así **CPython** está escrito en **C**, el **JPYthon** está en **java** e **IronPyton** está escrito en **C#**.
 - **CPython** es la más utilizada, la más rápida y la más madura. Cuando se habla de **Python** normalmente se habla de esta implementación. En este caso tanto el intérprete como los módulos están escritos en **C**.

Instalación de Python

- Para programar en **Python** basta con una versión del intérprete y un editor de texto simple.
- Nosotros instalaremos **Eclipse** con un complemento para el desarrollo bajo **Python** que nos proporcionará todas las facilidades modernas de programación como el sangrado de líneas resaltado, el autorelleno, la depuración, etc.

➤ Pasos:

1. Instalación del intérprete Python. descargaremos la versión **2.7**

(<http://www.python.org/getit/>) si vamos a desarrollar desde un entorno Windows, en caso de Linux no es necesario ya que la mayoría de las distribuciones incluyen los paquetes necesarios para su funcionamiento. Es importante saber el directorio en el que se instala ya que lo utilizaremos en la configuración de **Eclipse** si no lo detecta automáticamente.

2. Descarga e instalación de eclipse. Una que vez nos hayamos hecho con eclipse (<http://www.eclipse.org/downloads/>), la versión estándar nos puede servir, lo instalamos en nuestra máquina y probamos que funciona.



3. Instalación de PyDev para Eclipse. **PyDev** es un módulo desarrollado para Aptana que se puede usar dentro de **Eclipse** para programar bajo **Python**.

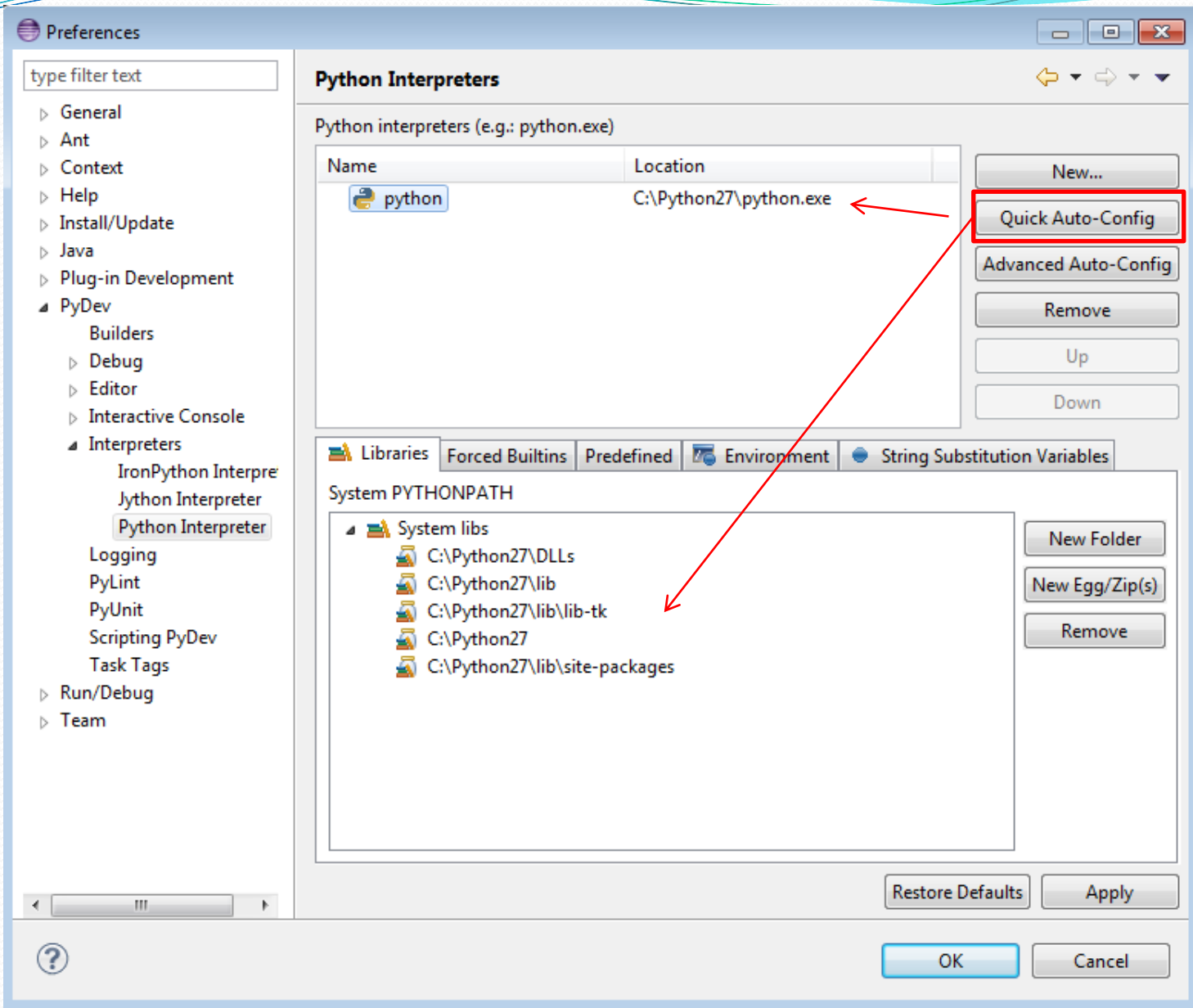
<http://comiendobocabytes.com/instalacion-de-pydev-en-eclipse/>

* Antes de proceder a instalar el plugin, actualiza la versión de java.

*Durante el proceso de instalación se mostrará un diálogo en el que se pide confirmación en la veracidad de un certificado. Es importante seleccionar y aceptar el valor que se nos propone. Si no lo hacemos tendremos que desinstalar el software y volver a comenzar.

4. Configuración de PyDev para eclipse. Abriendo las preferencias de **Eclipse** (menú **Window-Preferences**), en el árbol que muestra las categorías, en la parte izquierda de la ventana desplegaremos el elemento **PyDev**, buscando dentro la sub-categoría **Interpreters**. En esta opción debe configurarse la ruta del intérprete instalado en el paso uno. Generalmente pulsando el botón **Quick Auto-Config** debería detectar correctamente los valores necesarios. En caso de que no fuera así, abrían que añadirse a mano.

- En el siguiente cuadro de diálogo, vemos cómo ha localizado automáticamente el intérprete de **Python** instalado y ha completado los valores. Si tenemos alguna duda podemos consultar el enlace http://pydev.org/manual_101_interpreter.html.



Actividad 5.1

Descarga e instala el intérprete Python.

Actividad 5.2

Descarga e instala Eclipse.

Actividad 5.3

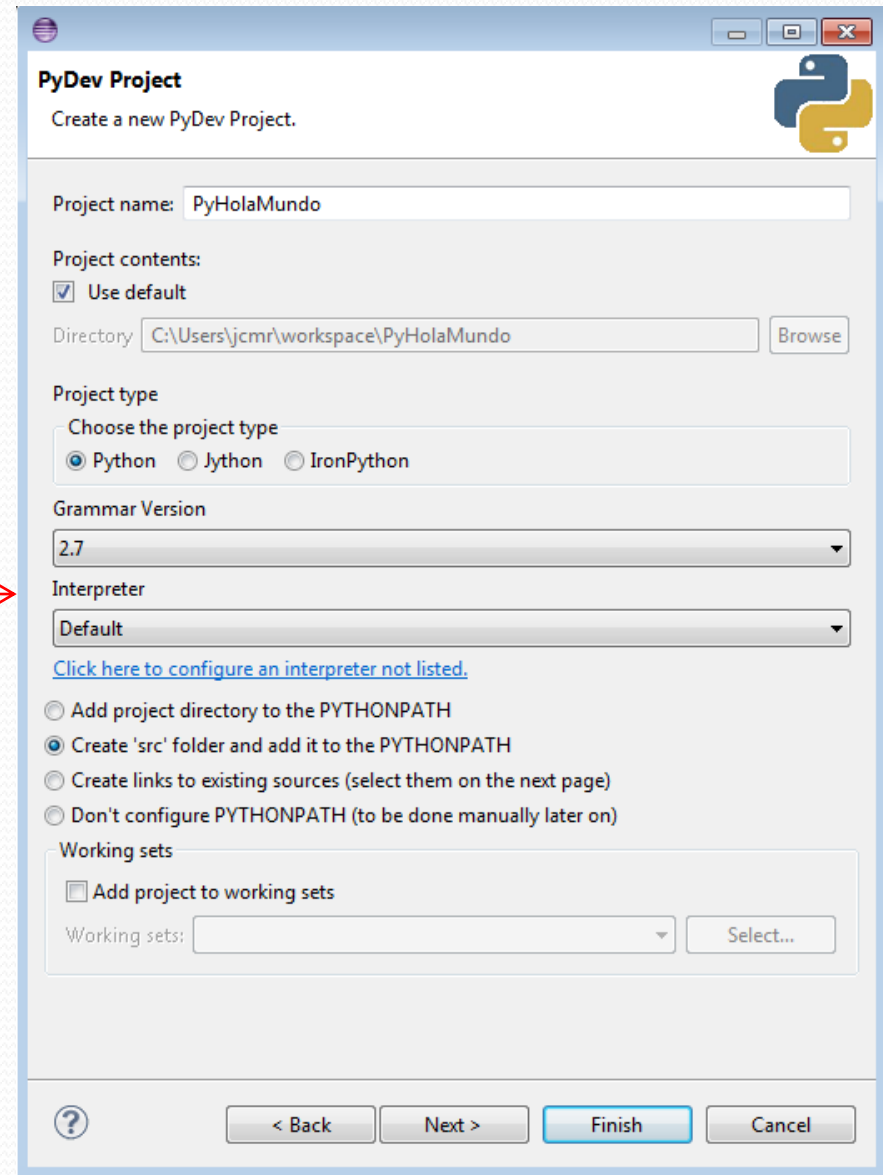
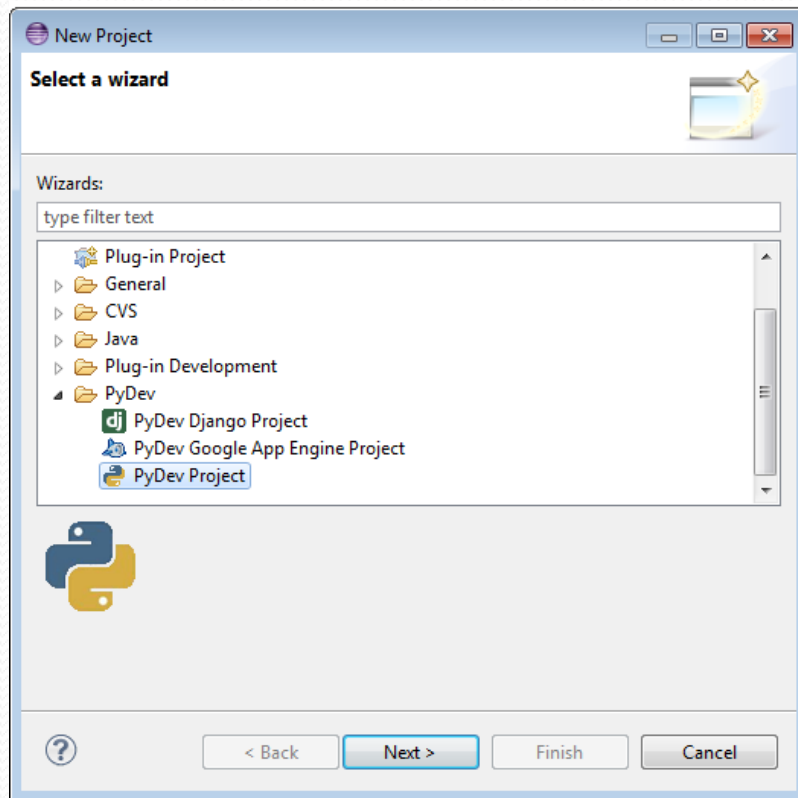
Instala PyDev para Eclipse.

Actividad 5.4

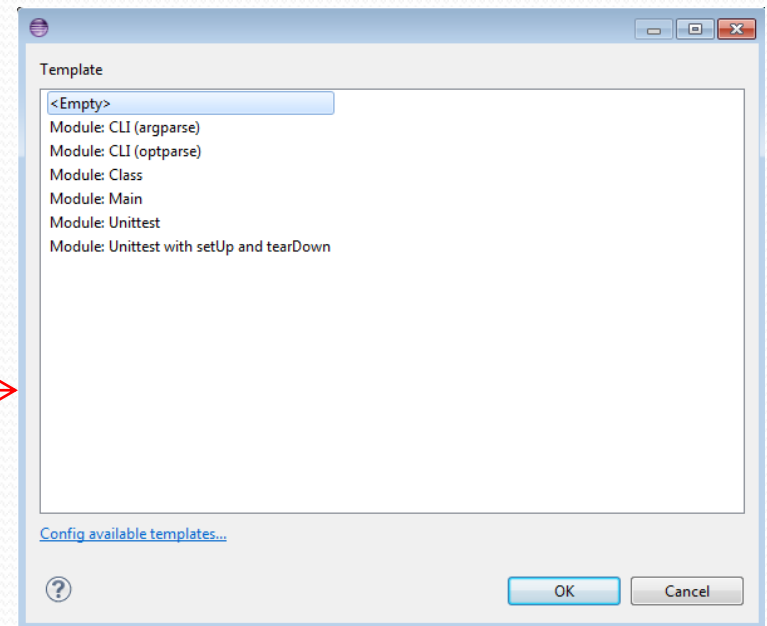
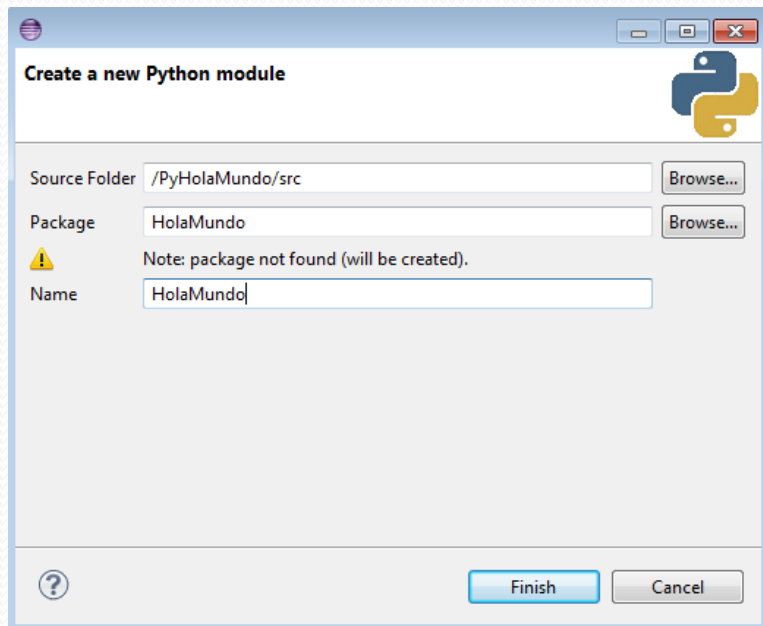
Configura el intérprete PyDev en Eclipse.

Crear un proyecto de prueba

- Vamos a comprobar que la configuración ha sido correcta creando un proyecto de prueba. Pasos:
 1. **Creemos un proyecto Python.** Pulsamos en **File- New-Project** y veremos el cuadro diálogo de la siguiente diapositiva, donde seleccionaremos **PyDev Project**.
 - Haremos clic en **Next** y estableceremos las propiedades como en la imagen siguiente. Es imprescindible asegurarse de que el nombre del proyecto está introducido, se selecciona el tipo de proyecto como **Python** y la versión 2.7. *Obligaremos también a que nos cree un subdirectorio por defecto para el código fuente llamado **src**.*

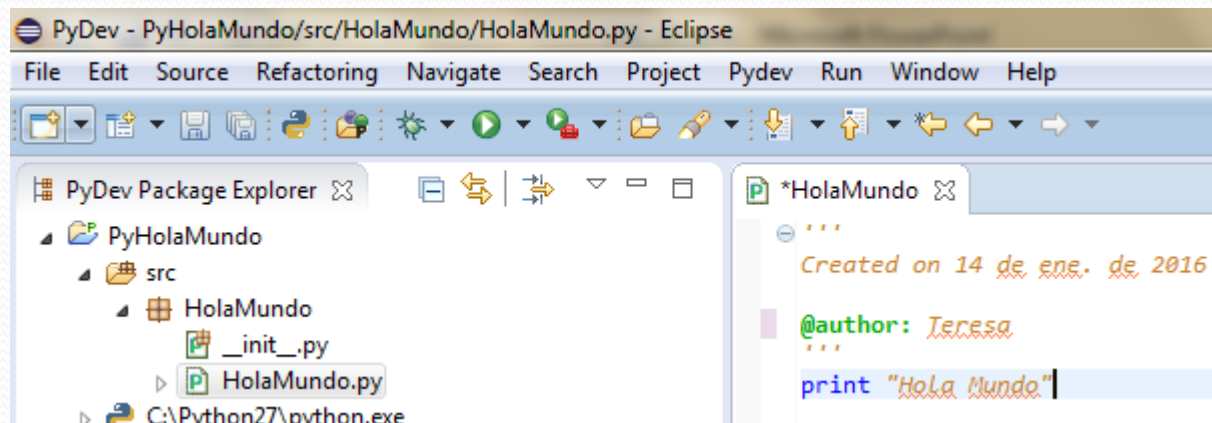


2. A continuación, añadimos un módulo abriendo el menú contextual sobre el directorio **src**, seleccionando la opción **New-PyDev Module**. En la nueva ventana que aparece fijaremos únicamente los valores de **Name**, en este caso a **HolaMundo**, en el campo **Template** seleccionaremos la opción **<Empty>**.



3. Creamos el código fuente. Desplegando el archivo holaMundo.py teclearemos:

```
print "Hola Mundo"
```



4. Guardamos el fichero y lo ejecutamos desde el botón **Run** de la barra de herramientas o desde el menú del mismo nombre. El diálogo nos pedirá el tipo de ejecución que debe lanzar, seleccionaremos **Python run** y en la consola situada en la parte inferior aparecerá el resultado.

Tipos básicos

- En Python los tipos básicos se dividen en:
 - Números, como pueden ser 3 (entero), 15.57 (de coma flotante) o $7 + 5j$ (complejo).
 - Cadenas de texto, como “Hola Mundo”.
 - Valores booleanos: True (cierto) y False (falso).

Números

- La representación numérica utiliza **enteros simples** (3) y **enteros largos** (3L) diferenciándose ambos en el valor máximo que pueden almacenar. Se puede utilizar la notación tradicional en **octal** (027) y **hexadecimal** (0x3F), **decimales o en coma flotante** (23.45) y **números complejos** ($2+7i$). Lo único que hay que tener en cuenta es que el valor máximo de cada tipo dependerá de la implementación del intérprete y la plataforma en la que se ejecute.

- Los números en **coma flotante** se utilizan para representar valores decimales, siendo la precisión mínima garantizada por **Python** la expresada en el estándar IEEE 754 donde se indica que los valores estarán comprendidos entre $\pm 2.22 \cdot 10^{308}$ y $\pm 1.79 \cdot 10^{308}$.
- Para aquellas situaciones en las que necesitemos mayor precisión usaremos **Decimal**. Este tipo es un objeto específico de **Python** que hay que importar y proporciona precisión de hasta veintiocho decimales.
- Un número **Decimal** se puede describir usando la notación científica si lo necesitamos (val=0.22e-2).
- El tipo **complejo** permite utilizar valores con parte real y parte imaginaria (2+3i).

Operadores

- Los operadores numéricos básicos son la **suma (+)**, **resta (-)**, **multiplicación (*)**, **división (/)**, **división entera (//)**, **exponenciación (**)** y **módulo (%)**. No existe diferencia entre la división (/) y la división entera (//) si ambos operandos son enteros.
- Para el tratamiento de bits se incorporan los siguientes operadores: operación **and (&)**, operación **or (|)**, operación **xor (^)**, operación **not (-)**, **desplazamiento a la izquierda («)** y **desplazamiento a la derecha (»)**.

Para Python todos los elementos son objetos, por lo que los tipos numéricos también lo son. Los números incorporan un conjunto de métodos que dan acceso a funciones matemáticas avanzadas como la raíz cuadrada (sqrt), logaritmos (log10), etc. **Eclipse** los muestra en cuanto tecleamos el punto en una variable con un valor entero.

Actividad 5.5

Crea un proyecto llamado **Numeros** y dentro un módulo llamado **Actividad5-5** con el código siguiente. Ejecútalo.

```
# -*- coding: utf-8 -*-
'''
Created on 14 de ene. de 2016

@author: Teresa
'''

# Esta línea es un comentario
# Numeros
iNum1 = 23 # Entero como int de C
fNum1 = 3.34 # float o reales usa doble precisión
INum1 = 3L # Entero largo como long de C
iNum2 = 027 # Octal
iNum3 = 0x3F # Hexadecimal
cNum1 = 2.1 + 3.1j # Complejos
cNum2 = -4.3 - 1j
print "Valores", iNum1, fNum1, cNum1
print "Suma de los dos primeros", iNum1 + fNum1
print "Suma de los dos últimos", fNum1 + cNum1
print "Suma de dos complejos", str(cNum1) + "+" + str(cNum2) + "=" + str(cNum1 + cNum2)
print "Octales y hexadecimales", iNum2, iNum3
```

Console

```
<terminated> C:\Users\Web\workspace\Numeros\src\Actividad5-5\Actividad5-5.py
Valores 23 3.34 (2.1+3.1j)
Suma de los dos primeros 26.34
Suma de los dos últimos (5.44+3.1j)
Suma de dos complejos (2.1+3.1j)+(-4.3-1j)=(-2.2+2.1j)
Octales y hexadecimales 23 63
```

Actividad 5.6

Dentro del proyecto **Numeros**, crea un módulo **Actividad5-6**. Ejecútalo.

```
from decimal import Decimal, getcontext
fNum1 = 3.34 # float o reales usa doble precision
dNum1 = Decimal(fNum1) # la mayor precision posible
print "real y decimal", fNum1, dNum1
getcontext().prec = 6
print "Precision 6:", Decimal(1) / Decimal(7)
getcontext().prec = 28
print "Precision 28:", Decimal(1) / Decimal(7)
```

Console

```
<terminated> C:\Users\Web\workspace\Numeros\src\Actividad5-6\Actividad5-6.py
real y decimal 3.34 3.339999999999999857891452847979962825775146484375
Precision 6: 0.142857
Precision 28: 0.1428571428571428571428571428571429
```

Actividad 5.7

Crea un proyecto llamado **operadores** y dentro un módulo llamado **Actividad5-7** con el código siguiente. Ejecútalo.

```
# Operadores +,-,*,/,//,**,%
from decimal import Decimal
iNum1 = 23 # Entero como int de C
fNum1 = 3.34 # float o reales usa doble precision
INum1 = 3L # Entero largo como long de C
print "multiplicacion:" + str(iNum1) + "*" + str(fNum1) + "=", iNum1 * fNum1
print "division:" + str(iNum1) + "/Decimal(" + str(fNum1) + ")=", iNum1 / Decimal(fNum1)
print "division:" + str(iNum1) + "/" + str(fNum1) + "=", iNum1 / fNum1
print "division entera:" + str(iNum1) + "//" + str(fNum1) + "=", iNum1 // fNum1
print "division (solo enteros):" + str(iNum1) + "/" + str(INum1) + "=", iNum1 / INum1 # Determina division entera
print "modulo:" + str(iNum1) + "%" + str(INum1) + "=", iNum1 % INum1
print "exponente:" + str(INum1) + "**" + str(INum1) + "=", INum1 ** INum1
```

Console

```
<terminated> C:\Users\Web\workspace\Operadores\src\Actividad5-7\Actividad5-7.py
multiplicacion:23*3.34= 76.82
division:23/Decimal(3.34)= 6.886227544910179933710117295
division:23/3.34= 6.88622754491
division entera:23//3.34= 6.0
division (solo enteros):23/3= 7
modulo: 23%3= 2
exponente:3**3= 27
```

Actividad 5.8

Dentro del proyecto **Operadores**, crea un módulo **Actividad5-8**.. Ejecútalo.

```
# Operadores a nivel de bit:
bNum1 = 0x9 #1001
bNum2 = 0xC #1100
print "And (&)", bNum1 & bNum2 # 1000
print "Or (|)", bNum1 | bNum2 # 1101
print "XOR (^)", bNum1 ^ bNum2 # 0101
print "Not(~)", ~bNum1
print "Desplazamiento >> de 2", bNum1 >> 2 # 0010
print "Desplazamiento << de 2", bNum1 << 2 # 0010 0100 tener en cuenta que son 32 o 64 bits la representacion
```

Console

<terminated> C:\Users\Web\workspace\Operadores\src\Actividad5-8\Actividad5-8.py

```
And (&) 8
Or (|) 13
XOR (^) 5
Not(~) -10
Desplazamiento >> de 2 2
Desplazamiento << de 2 36
```


Actividad 5.9

Dentro del proyecto **Numeros**, crea un módulo **Actividad5-9**. Ejecútalo.

```
from decimal import Decimal
fNum1 = 3.34 # float o reales usa doble precision
INum1 = 3L # Entero largo como long de C
dNum1 = Decimal(fNum1) # la mayor precision posible
print "    Funciones basicas    "
print dNum1 .copy_abs()
print dNum1 .is_infinite()
print dNum1.log10()
print dNum1.sqrt()
print dNum1.max(INum1)
print fNum1.hex() # representacion hexadecimal
print INum1.bit_length() # bits necesarios para representar el valor
```

Console

<terminated> C:\Users\Web\workspace\Numeros\src\Actividad5-9\Actividad5-9.py

```
    Funciones basicas
3.339999999999999857891452847979962825775146484375
False
0.5237464668115644567196837090
1.827566688249706486724138495
3.339999999999999857891452848
0x1.ab851eb851eb8p+1
2
```

Booleanos

- Se utiliza para describir expresiones condicionales, pudiendo almacenar exclusivamente los valores **True** o **False**.
- Para este tipo se definen operandos especiales de comparación: comparación y (and), comparación o (or), negación de la expresión (not), junto con los tradicionales de igualdad (==), desigualdad (!=), mayor (>), mayor o igual (>=), menor (<) y menor o igual (<=).

Operador	Descripción	Ejemplo
and	¿se cumple a y b?	<code>r = True and False # r es False</code>
or	¿se cumple a o b?	<code>r = True or False # r es True</code>
not	No a	<code>r = not True # r es False</code>

Operador	Descripción	Ejemplo
==	¿son iguales a y b?	<code>r = 5 == 3 # r es False</code>
!=	¿son distintos a y b?	<code>r = 5 != 3 # r es True</code>
<	¿es a menor que b?	<code>r = 5 < 3 # r es False</code>
>	¿es a mayor que b?	<code>r = 5 > 3 # r es True</code>
<=	¿es a menor o igual que b?	<code>r = 5 <= 5 # r es True</code>
>=	¿es a mayor o igual que b?	<code>r = 5 >= 3 # r es True</code>

Actividad 5.10

Crea un proyecto llamado **Booleanos** y dentro un módulo llamado **Actividad5-10** con el código siguiente. Ejecútalo.

```
# -*- coding: utf-8 -*-
'''
Created on 14 de ene. de 2016

@author: Teresa
'''

# Booleanos
# Operadores and, or y not
print "True and True:", True and True
print "True and False:", True and False
print "True or True:", True or True
print "True or False:", True or False
print "Not True:", not True
# Operadores de comparación: ==, !=, <, <=, >, >=
print "'a'=='a'", 'a' == 'a'
print "'a'=='b'", 'a' == 'b'
```

Console

```
<terminated> C:\Users\Web\workspace\Booleanos\src\Actividad5-10\Actividad5-10.py
True and True: True
True and False: False
True or True: True
True or False: True
Not True: False
'a'=='a' True
'a'=='b' False
```

Cadenas

- Podemos representar una cadena mediante **comillas simples** o mediante comillas **dobles**, pero nunca mezclando notaciones.
- Dentro de ambas notaciones es posible introducir valores especiales mediante la barra invertida (\). Así `\n` representará el carácter de nueva línea, `\t` el de tabulación, `\r` el de retroceso de carro, etc.
- Si no deseamos que la barra invertida se interprete junto con el siguiente carácter antepondremos `r` a la definición de la cadena fuera de las comillas (`r"\na\t"`).
- Si nuestra cadena está formada por valores UTF-8 usaremos `u` como carácter predecesor.

- Un mecanismo nuevo de definición de cadenas son **las tres comillas dobles al inicio y fin de la definición**. De esta forma podremos escribir el **texto en varias líneas**, y al imprimir la cadena, se respetarán los saltos de línea que introdujimos sin tener que recurrir al carácter `\n`. Ejemplo:

```
triple = """primera linea  
esto se vera en otra linea"""
```

- Para que el archivo se guarde en formato **UTF-8** permitiendo caracteres acentuados y demás signos españoles, pondremos al principio del fichero la línea

```
# -*- coding: utf-8 -*-
```

- Las cadenas también admiten operadores como **+**, que funciona realizando una **concatenación** de las cadenas utilizadas y *****, en la que se **repite la cadena** tantas veces como lo indique el número utilizado como segundo operando.

➤ Las cadenas son también objetos, por lo que se incorporan algunas **funciones útiles**:

- **capitalize()** devolverá una cadena con la primera letra en mayúsculas,
- **center(num)** centrará la cadena en el nº de caracteres indicado usando blancos,
- **ljust()** y **rjust()** justifican a la izquierda y derecha respectivamente,
- **count(subcadena)** devuelve el nº de ocurrencias de la cadena pasada como parámetro,
- **find(subcadena)** devuelve la 1ª posición en la que aparece la cadena del parámetro,
- **upper()** convierte a mayúsculas,
- **strip()** elimina los blancos en los extremos de la cadena,
- **split(carácter)** divide la cadena en partes utilizando como separador el carácter pasado, **splitlines()** divide según líneas,
- **len(cadena)** longitud del objeto, esta función se puede usar con cualquier objeto para encontrar su tamaño,
- **join(cadena)** hace una unión de las dos
- **format(valores)** formatea la cadena de entrada según la expresión contenida con los valores que se pasan.

Actividad 5.11

Crea un proyecto llamado **Cadenas** y un módulo llamado **actividad5-11** con el código siguiente. Ejecútalo.

```
# -*- coding: utf-8 -*-
sCad1 = "Mi cadena 'ñ'\tes esta" # la ñ aparece bien por la primera línea del código fuente
sCad2 = 'Mi Cadena "ñ"\tes esta'
sCad3 = r"Mi Cadena \tñ'es esta" # raw
sCad4 = u"Mi cadena 'ñ'\tes esta" # unicode
sCad5 = ""
Mi Cadena \t'ñ'es esta""
sCad6 = "Valor:"
sCad7 = "esta frase es de prueba"
sCad8 = "-";
sCad9 = ("a", "b", "c"); # Arrays de cadenas, los veremos más adelante
sCad10 = "El valor de {} + {} es {}" # Las llaves denotan posición de los parámetros a sustituir 0,1,2
# Las llaves se escapan duplicándolas ver http://docs.python.org/2/library/string.html#formatstrings
print "Entre comillas dobles:", sCad1
print "Entre comillas simples:", sCad2
print "Cadena raw:", sCad3
# Funciones básicas
print "Funciones básicas"
print sCad7.capitalize()
print sCad7.center(50)
print sCad7.ljust(50)
print sCad7.rjust(50)
print sCad7.count("es")
print sCad7.find("se")
print sCad7.upper()
print sCad7.strip()
print sCad7.split(" ")
print sCad7.splitlines()
print len(sCad7)
print sCad8.join(sCad9);
print sCad10.format(1, 2, 1 + 2)
```


Actividad 5.11 (...continuación)

Console

<terminated> C:\Users\Web\workspace\Cadenas\src\Actividad5-11\Actividad5-11.py

Entre comillas dobles: Mi cadena 'ñ' es esta

Entre comillas simples: Mi Cadena "ñ" es esta

Cadena raw: Mi Cadena \tñ'es esta

Funciones básicas

Esta frase es de prueba

esta frase es de prueba

esta frase es de prueba

esta frase es de prueba

2

8

ESTA FRASE ES DE PRUEBA

esta frase es de prueba

['esta', 'frase', 'es', 'de', 'prueba']

['esta frase es de prueba']

23

a-b-c

El valor de 1 + 2 es 3

Diccionarios o tablas hash

- Los **diccionarios**, también llamados **matrices asociativas** o tablas **hash**, deben su nombre a que son *colecciones que relacionan una clave y un valor*.
- La definición de un diccionario se realiza mediante llaves «{}», separando cada registro por una coma «,» y la clave de su valor por dos puntos «:». Por ejemplo, un diccionario de películas y directores:

d = {"Gladiator": "Ridley Scott", "Kill Bill": "Quentin Tarantino",
"Titanic": "Cameron"}
- Una vez creada tendremos acceso a cada uno de los elementos mediante la notación de corchetes (**dVal["clave"]**), donde el valor a utilizar dentro será el nombre o valor de la clave usado en la definición, nunca una posición.

- Se utilizarán claves de los tipos básicos numéricos o cadenas teniendo en cuenta que la clave es sensible a la capitalización, ningún otro es aceptado siendo posible mezclar los tipos de las claves en un mismo diccionario o en los datos.
- La clave no se podrá repetir ni variar una vez creada, pero su contenido se puede modificar en cualquier momento usando los corchetes (**dVal["clave"]=valor**). Por ejemplo:

dval["Titanic"] = "James Cameron"

- Si necesitamos extender el diccionario, lo realizaremos usando la notación de corchetes y una clave que no hayamos usado, el borrado de un registro es tan sencillo como utilizar la función **del** con el registro a borrar (**del dVal["clave"]**) o usar el método **clear()** para borrarla completamente.

Actividad 5.12

Crea un proyecto llamado **Diccionarios** y un módulo llamado **Actividad5-12** con el código siguiente. Ejecútalo.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-



#Definición
dVal1={"key1":"valor1", "key2":"valor2"}
print "Diccionario:", dVal1
print "Diccionario valor con clave key2 d['key2']:", dVal1['key2']

#Se puede usar cualquier tipo de valor, y en las claves tipos básicos
dVal2 = {1:"Valor cadena","keyABC":23}
print "Diccionario:", dVal2
print "Diccionario valor con clave 1 d[1]:", dVal2[1]
print "Diccionario valor con clave keyABC d['keyABC']:", dVal2['keyABC']

#Es dinámico pero sin claves duplicadas
dVal2[2] = 6 + 7j
print "Diccionario añadir:", dVal2
dVal2[2] = "ABC"
print "Diccionario modificar:", dVal2
del dVal2[1]
print "Diccionario borrar:", dVal2

#Funciones básicas
print "Funciones básicas"
print len(dVal2)
print dVal2.items() #Lista de todo
print dVal2.keys() #Lista de las claves
print dVal2.values() #Lista de los valores
print dVal2.has_key("1") # Existe la clave 1. Es una cadena de texto debe dar falso
dVal2.clear()
```

Actividad 5.12 (...continuación)

 Console 

```
<terminated> C:\Users\Web\workspace\Diccionarios\src\Actividad5-12\Actividad5-12.py
```

```
Diccionario: {'key2': 'valor2', 'key1': 'valor1'}
```

```
Diccionario valor con clave key2 d['key2']: valor2
```

```
Diccionario: {1: 'Valor cadena', 'keyABC': 23}
```

```
Diccionario valor con clave 1 d[1]: Valor cadena
```

```
Diccionario valor con clave keyABC d['keyABC']: 23
```

```
Diccionario añadir: {1: 'Valor cadena', 2: (6+7j), 'keyABC': 23}
```

```
Diccionario modificar: {1: 'Valor cadena', 2: 'ABC', 'keyABC': 23}
```

```
Diccionario borrar: {2: 'ABC', 'keyABC': 23}
```

```
    Funciones básicas
```

```
2
```

```
[(2, 'ABC'), ('keyABC', 23)]
```

```
[2, 'keyABC']
```

```
['ABC', 23]
```

```
False
```