

# **Jython**

Antonio Espín Herranz

# Contenidos














- Introducción
- Instalación de Java
- Instalación de Jython
- Instalación de Eclipse
- Configurar proxy
- Instalación de PyDev: plugin de eclipse para Python
- Configuración plugin: Pydev
- Importación de Librerías Java
- Añadir jars
- Colecciones
- PythonInterpreter

# Jython

- Jython (Python en Java), es una implementación del lenguaje Python para la plataforma Java.
- Desde jython podemos utilizar clases de Java.
- El código se escribe con el lenguaje python pero podemos utilizar clases de Java.
- Y también desde Java podemos interactuar con el intérprete de Jython.
- Página principal del proyecto:
  - <http://www.jython.org/>

# Instalación de Java

- Descargar la JDK de la página de Oracle.
  - Por ejemplo con la JDK 8
  - Aceptar la licencia:
  - Descargar e instalar
  - <https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

Java SE Development Kit 8u191		
You must accept the <b>Oracle Binary Code License Agreement for Java SE</b> to download this software.		
<input type="radio"/> Accept License Agreement <input checked="" type="radio"/> Decline License Agreement		
Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	72.97 MB	 <a href="#">jdk-8u191-linux-arm32-vfp-hflt.tar.gz</a>
Linux ARM 64 Hard Float ABI	69.92 MB	 <a href="#">jdk-8u191-linux-arm64-vfp-hflt.tar.gz</a>
Linux x86	170.89 MB	 <a href="#">jdk-8u191-linux-i586.rpm</a>
Linux x86	185.69 MB	 <a href="#">jdk-8u191-linux-i586.tar.gz</a>
Linux x64	167.99 MB	 <a href="#">jdk-8u191-linux-x64.rpm</a>
Linux x64	182.87 MB	 <a href="#">jdk-8u191-linux-x64.tar.gz</a>
Mac OS X x64	245.92 MB	 <a href="#">jdk-8u191-macosx-x64.dmg</a>
Solaris SPARC 64-bit (SVR4 package)	133.04 MB	 <a href="#">jdk-8u191-solaris-sparcv9.tar.Z</a>
Solaris SPARC 64-bit	94.28 MB	 <a href="#">jdk-8u191-solaris-sparcv9.tar.gz</a>
Solaris x64 (SVR4 package)	134.04 MB	 <a href="#">jdk-8u191-solaris-x64.tar.Z</a>
Solaris x64	92.13 MB	 <a href="#">jdk-8u191-solaris-x64.tar.gz</a>
Windows x86	197.34 MB	 <a href="#">jdk-8u191-windows-i586.exe</a>
Windows x64	207.22 MB	 <a href="#">jdk-8u191-windows-x64.exe</a>

# Instalación de Java

- Una vez instalado, configurar las variables de entorno:
  - **JAVA\_HOME**
    - Carpeta de instalación
    - C:\Program Files\Java\jdk1.8.0\_65 (ejemplo)
  - **CLASSPATH**
    - %JAVA\_HOME%\lib\tools.jar;.
  - **PATH** (*se añade al PATH existente*)
    - ... ;%JAVA\_HOME%\bin; ...

# Instalación Jython

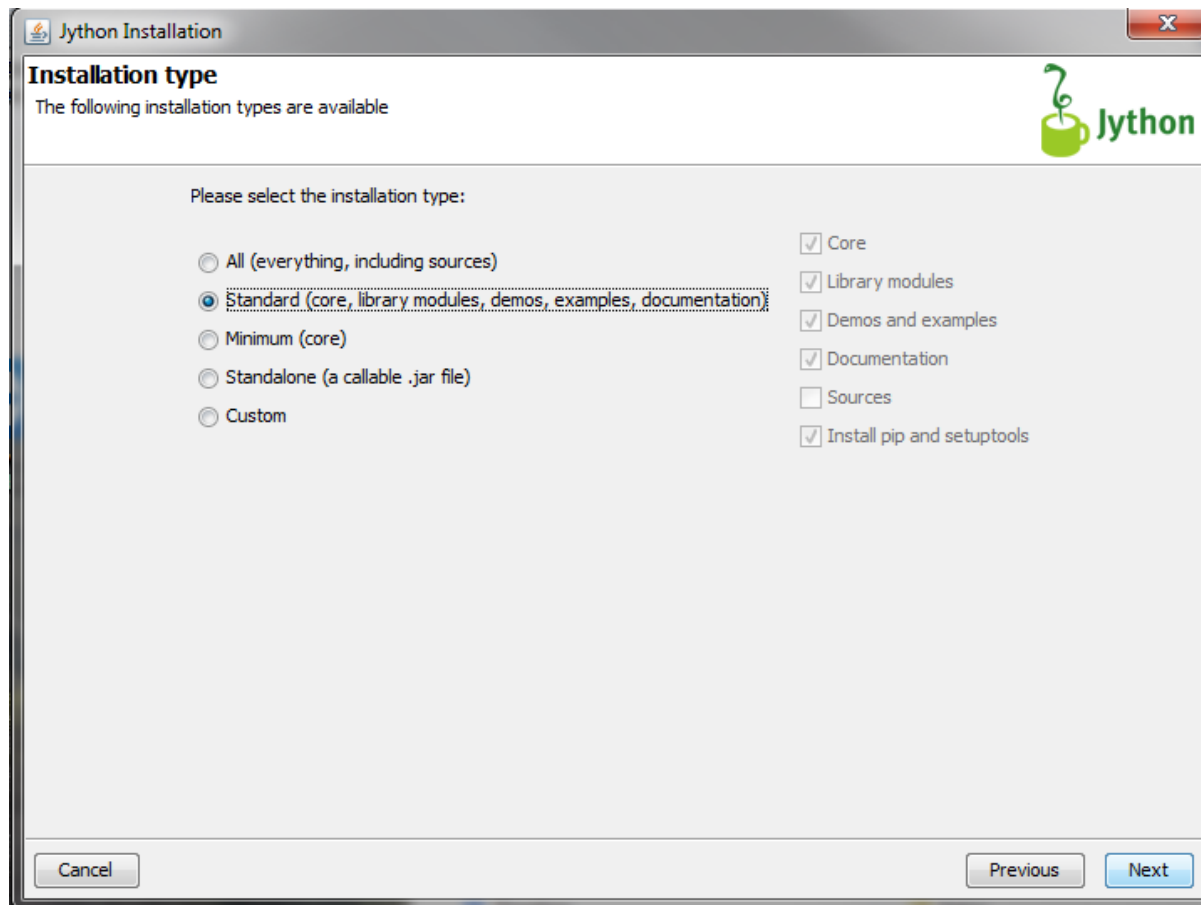
- Descargar el jar: <http://www.jython.org/downloads.html>
- Doble click en el fichero: **jython-installer-2.7.0.jar**



*El otro jar: jython-standalone-2.7.0.jar es para integrar Jython en aplicaciones Java*

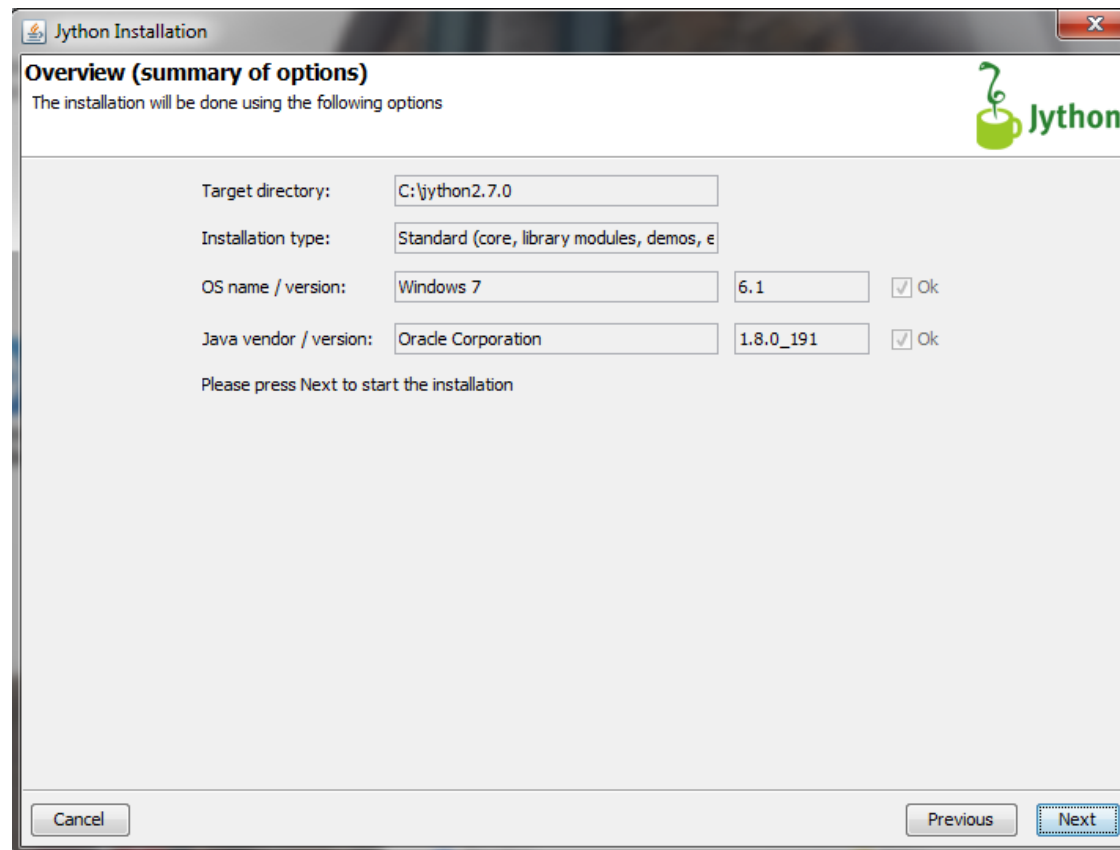
# Instalación de Jython

- Aceptar la licencia y seleccionar instalación estándar.



# Instalación de Jython

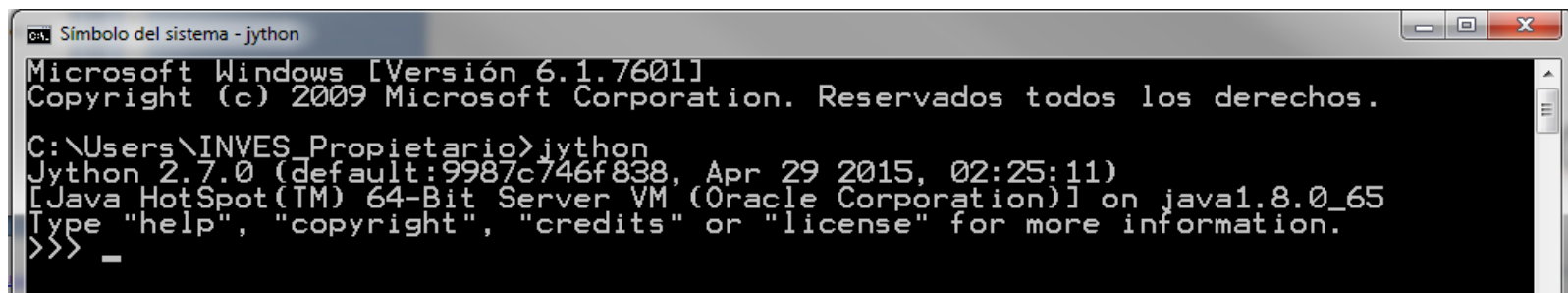
- Seleccionar el directorio destino (dejar el dir. Por defecto).
- En la siguiente pantalla muestra información del S.O. y la versión de Java (importante para que lo encuentre automáticamente que estén bien configuradas las var. De entorno)





# Instalación de Jython

- Cuando ha terminado la instalación, se pueden configurar unas **variables de entorno** para que sea más cómodo entrar en la **consola de Jython**.
- **JYTHON\_HOME**
  - C:\jython2.7.0
- **PATH** (se añade)
  - %JYTHON\_HOME%\bin
- En una consola: **jython**



```
Símbolo del sistema - jython
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Users\INVES_Propietario>jython
Jython 2.7.0 (default:9987c746f838, Apr 29 2015, 02:25:11)
[Java HotSpot(TM) 64-Bit Server VM (Oracle Corporation)] on java1.8.0_65
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

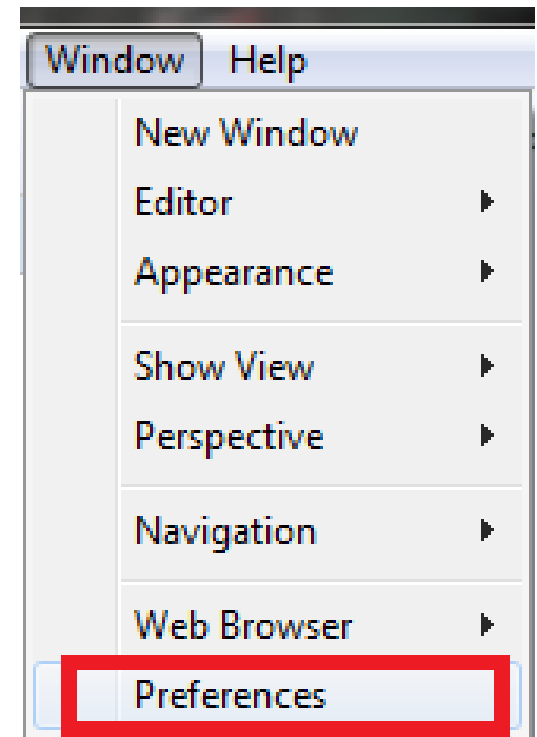
# Instalación de Eclipse

- Descargar el instalador:
  - Abrirá una ventana para descargar un eclipse.
  - Seleccionar la primera opción:
  - <https://www.eclipse.org/downloads/download.php?file=/oomph/epp/2018-09/Ra/eclipse-inst-win64.exe>

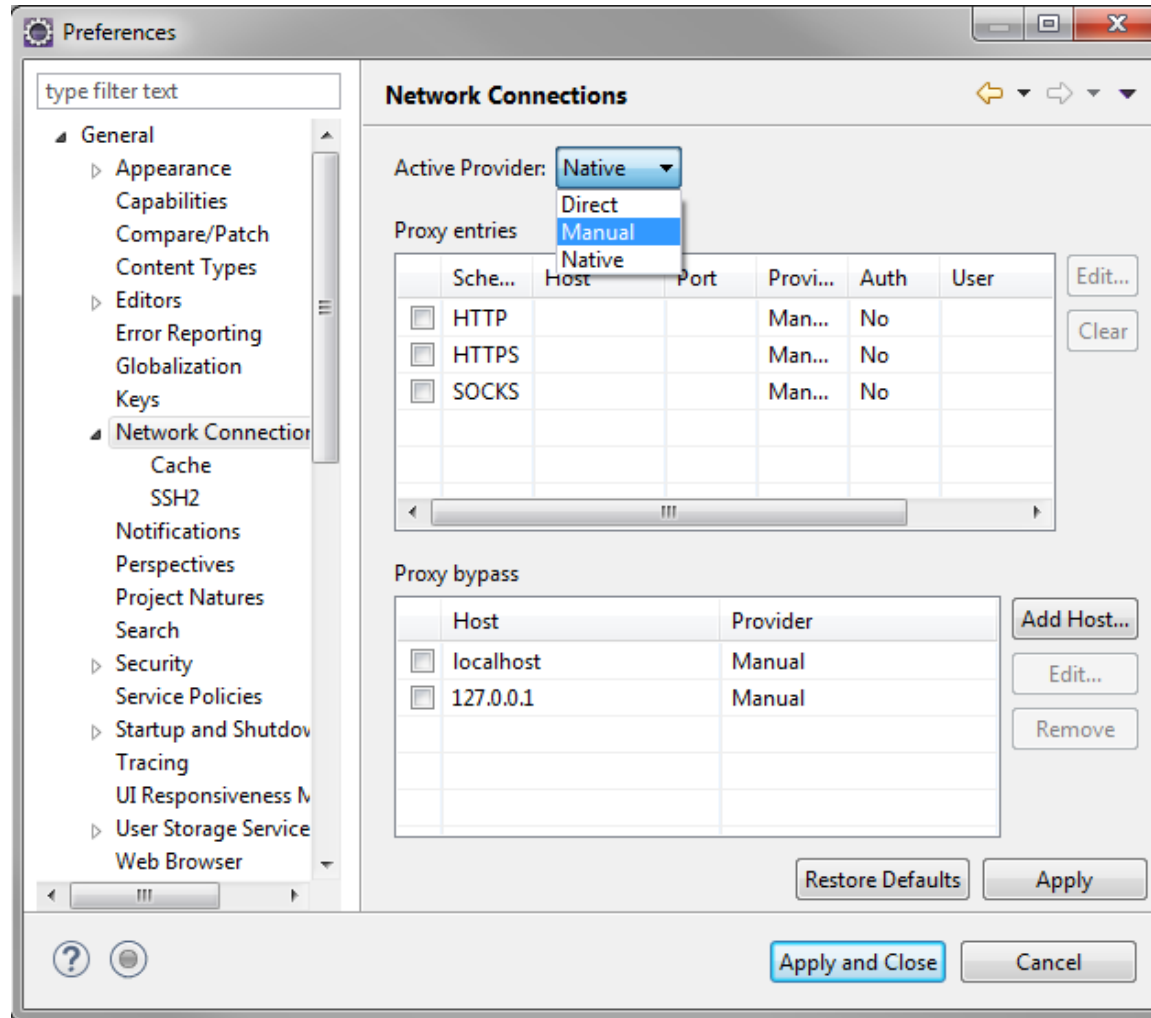


# Configurar Proxy

- Si es necesario configurar el proxy en eclipse para la instalación de plugins.
- En el menú principal:
  - Window → Preferences

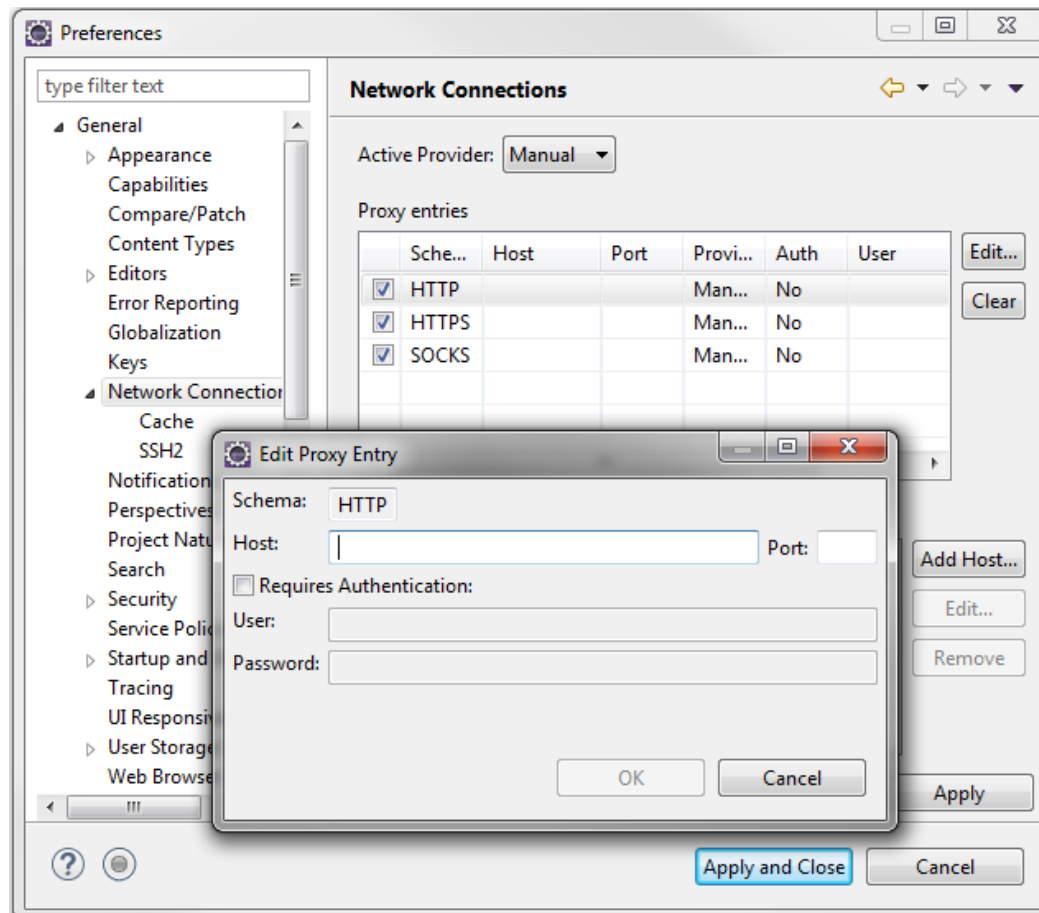


# Configurar Proxy



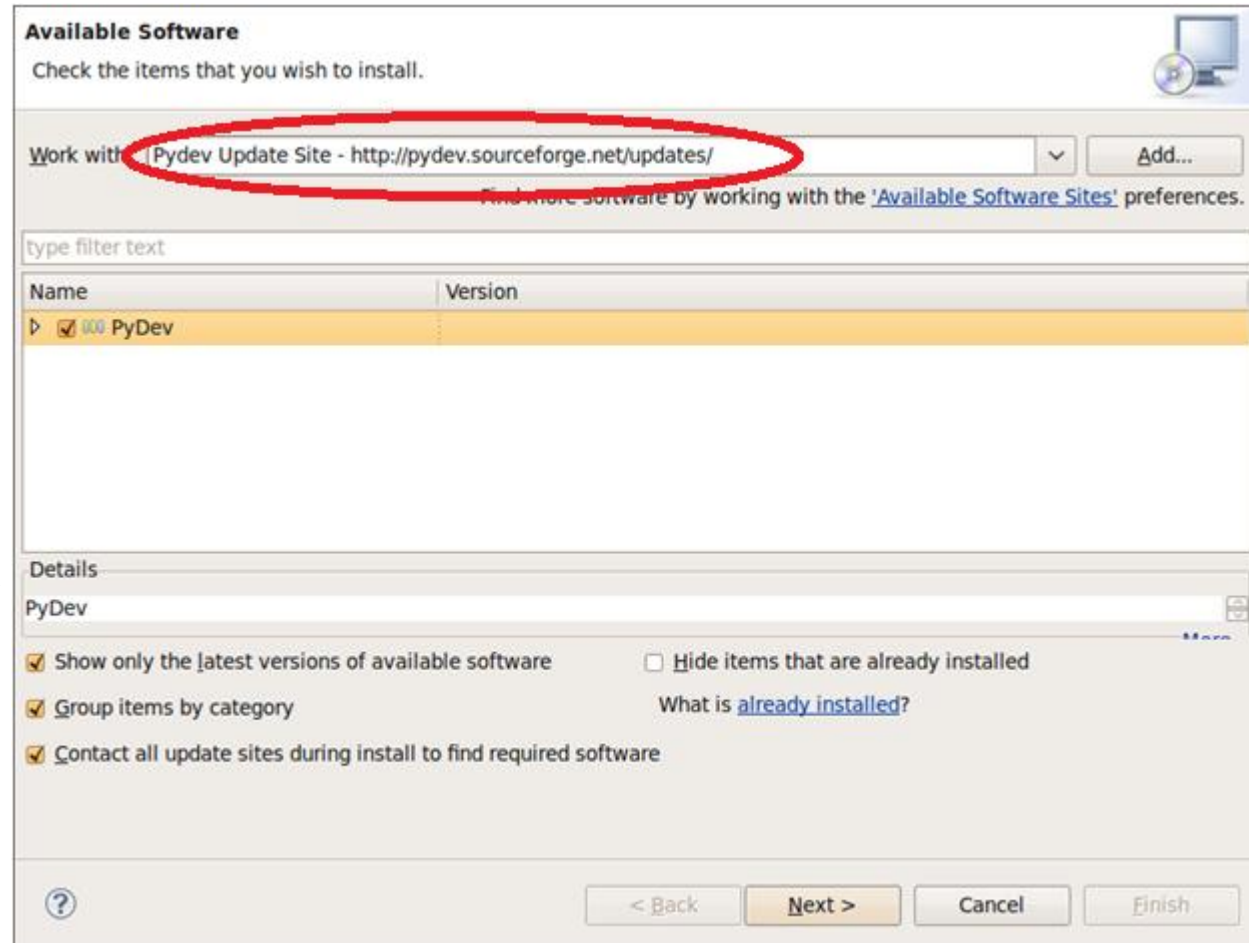
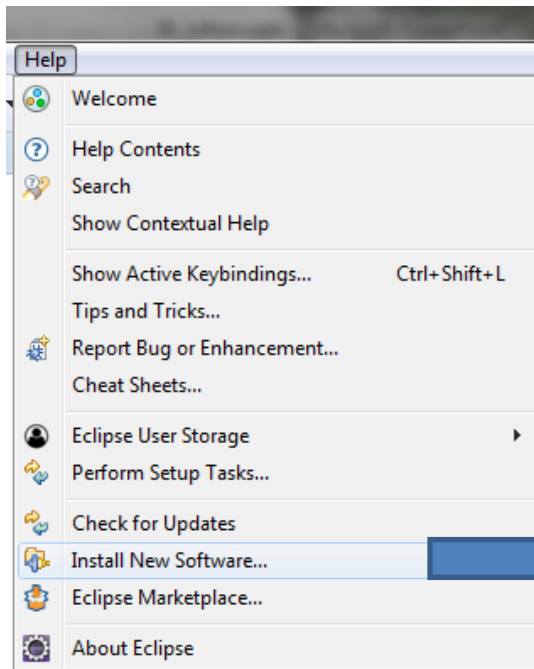
# Configurar Proxy

- Seleccionar HTTP, pulsar botón Edit.
- Añadir los datos del proxy.



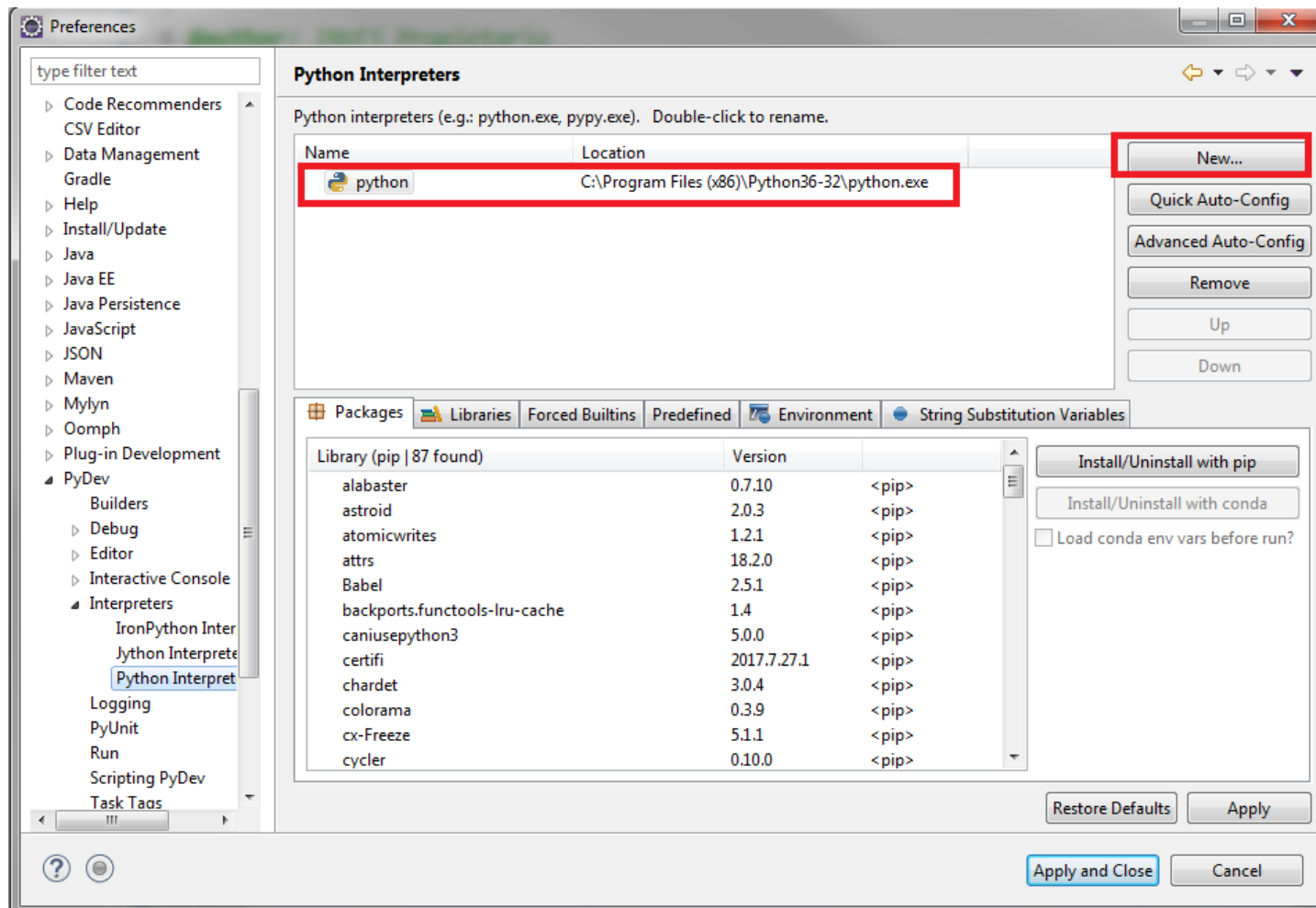
# Instalación Pydev

*En el menú principal → Help → Install New Software*



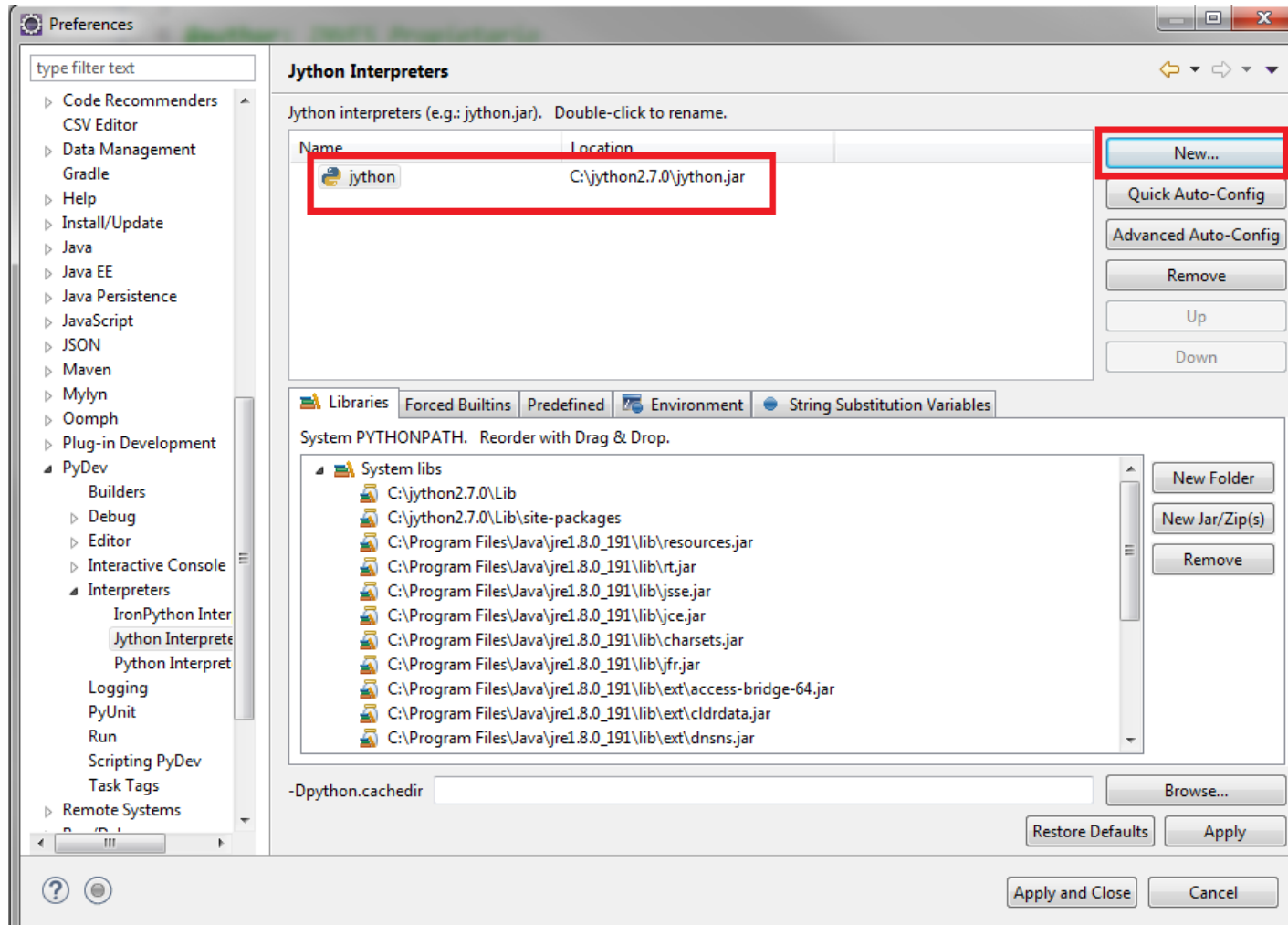
# Configurar Pydev

- Desde el menú principal → window → preferences:
  - ***Seleccionar el interprete de Python***



# Configurar Pydev

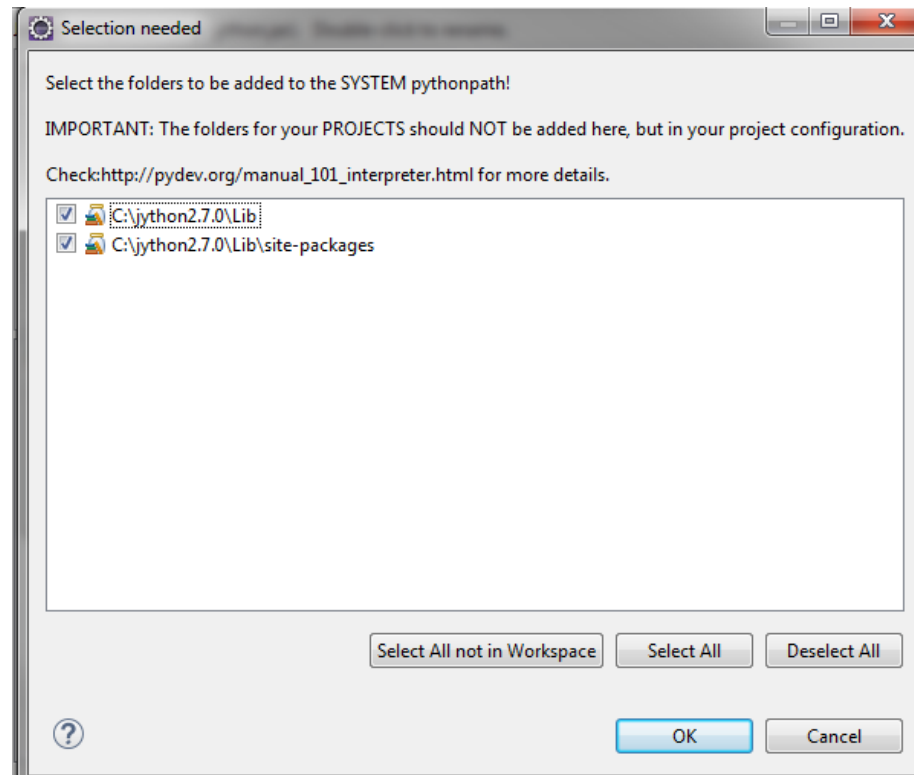
- De una forma similar seleccionar el interprete de jython.
- En este caso hay que seleccionar el archivo: **jython.jar**





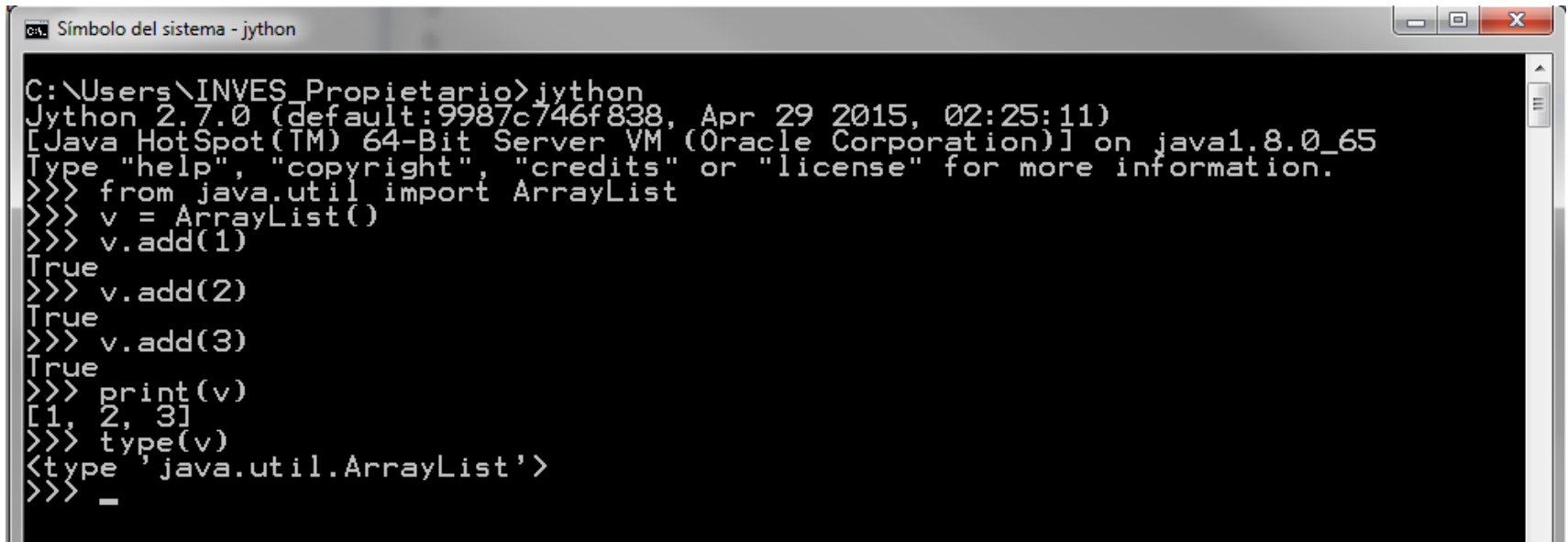
# Configurar Pydev

- Cuando se selecciona el fichero jar de jython, se mostrará una ventana similar a esta para indicar que hay algunas carpetas de la instalación jython se añadirán al PATH:



# Consola de Jython

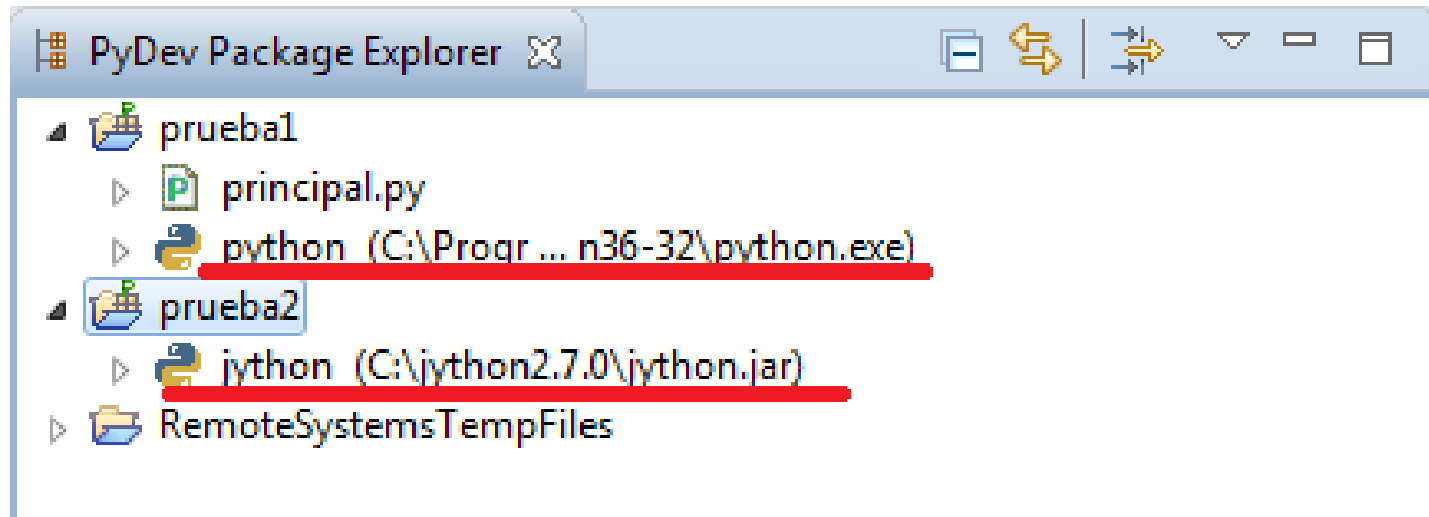
- Podemos interactuar con la consola de Jython de la misma forma que lo hacemos con la de python, utilizando clases de Java.



```
Símbolo del sistema - jython
C:\Users\INVES_Propietario>jython
Jython 2.7.0 (default:9987c746f838, Apr 29 2015, 02:25:11)
[Java HotSpot(TM) 64-Bit Server VM (Oracle Corporation)] on java1.8.0_65
Type "help", "copyright", "credits" or "license" for more information.
>>> from java.util import ArrayList
>>> v = ArrayList()
>>> v.add(1)
True
>>> v.add(2)
True
>>> v.add(3)
True
>>> print(v)
[1, 2, 3]
>>> type(v)
<type 'java.util.ArrayList'>
>>> _
```

# Crear proyectos

- Desde eclipse podemos crear proyectos y seleccionar el interprete que queremos utilizar:



# Ejemplo: Uso de clases Java - Jython

```
from java.util import ArrayList
from java.util import Random
```

```
rd = Random()
v = ArrayList()
```

```
for i in range(25):
    valor = rd.nextInt()
    v.add(valor)
```

```
print(v)
print(type(v))
```

# Tipos Java vs Python

Java	Python
char	String (must have length 1)
boolean	Integer (false = zero, true = nonzero)
byte, short, int,	Integer
long	Long (in the range of Java long or integer)
float, double	Float
java.lang.String	String
byte[]	String
array[]	Jarray

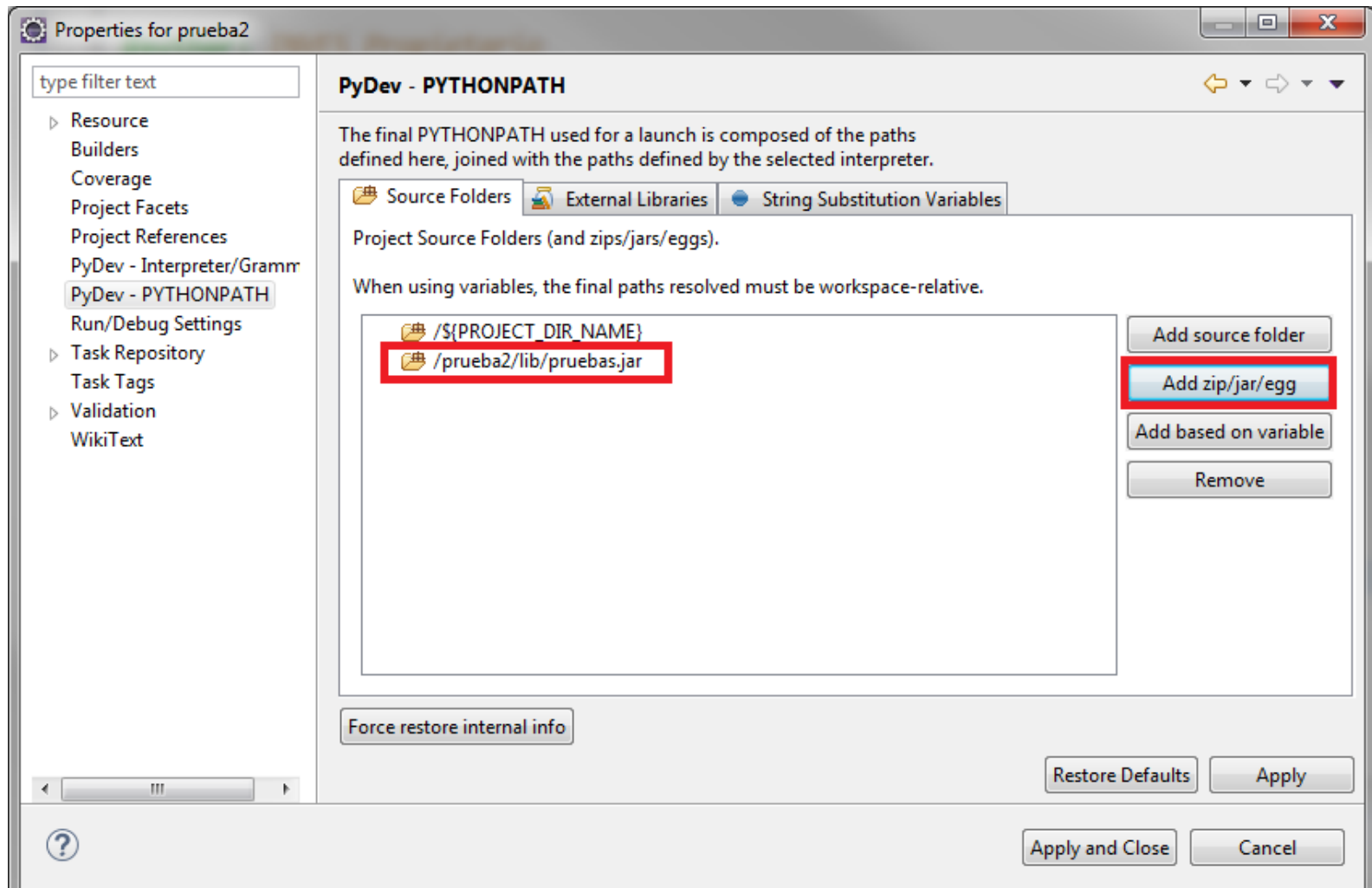
# Importar librerías

- Las librerías de java las importaremos utilizando la sintaxis de python.
- Pero con los paquetes de Java.
  - java.util
  - java.net
  - java.io
  - javax.swing
  - Etc.
- Tener en cuenta que si queremos trabajar con String (de Java) u otras clases básicas que se encuentran en el paquete java.lang de Java las tendremos que importar.  
**from java.lang import String**

# Añadiendo Jars

- Podemos **añadir ficheros jar** (java archive) desarrollados por nosotros o terceros a nuestro proyecto de Jython y después utilizarlos de la misma forma que las clases de Java.
- Dentro de nuestro proyecto **creamos una carpeta (lib, por convención), dentro copiamos todos los jars.**
- Con el **botón derecho sobre el proyecto**, seleccionar **propiedades → PYTHONPATH** (*ver siguiente página*)

# Añadiendo Jars





# Colecciones: Java

- Dentro de Java, tenemos varios tipos de colecciones, representadas por una serie de interfaces :
  - **List**
    - Listas dinámicas, lo más parecido a los arrays.
    - Índices numéricos, empiezan en 0.
    - Admiten repetidos.
    - Equivalente en python: list()
  - **Map**
    - Mapas: Arrays asociativos. Pares de clave:valor. No admiten repetidos.  
Equivalente: dict()
  - **Set**
    - Conjuntos. No admiten repetidos. Las claves son numéricas.
- En el código tenemos que trabajar con las implementaciones.

# Implementación colecciones: Java

- Con la interface **List**:
  - class **ArrayList**
    - Es una implementación muy eficiente en cuanto a uso de memoria.
    - Es rápida en todas las operaciones, excepto en las que afectan a elementos intermedios: inserción y borrado.
    - Puede decirse que es un “array” de tamaño dinámico.
  - class **LinkedList**
    - Es una implementación basada en listas encadenadas.
    - Esto ofrece una buena velocidad en operaciones sobre términos intermedios (inserción y borrado) a cambio de ralentizar las demás operaciones.
  - class **Vector**
    - Similar a “ArrayList” pero con métodos sincronizados, lo que permite ser usada en programas concurrentes.
    - Todo es más lento que con una ArrayList.

# Métodos

- `boolean add(E elemento)` : añade un elemento al final de la lista.
- `void add(int posicion, E elemento)` : inserta un elemento en la posición indicada.
- `void clear()` : vacía la lista.
- `boolean contains(E elemento)`: true si hay en la lista un elemento “equals” al indicado.
- `boolean equals(Object x)`: una lista es igual a otra si contienen en las mismas posiciones elementos que son respectivamente “equals”.
- `E get(int posicion)`: devuelve el elemento en la posición indicada.
- `int indexOf(E elemento)` : devuelve la posición en la que se haya el primer elemento “equals” al indicado; o -1 si no hay ningún elemento “equals”.
- `boolean isEmpty()`: true si no hay elementos.

# Métodos II

- `Iterator <E> iterator()` : devuelve un iterador sobre los elementos de la lista.
- `E remove(int posicion)`: elimina el elemento en la posición indicada, devolviendo lo que elimina.
- `boolean remove(E elemento)` : elimina el primer elemento de la lista que es “equals” el indicado; devuelve true si se elimina algún elemento.
- `E set(int posicion, E elemento)` : reemplaza el elemento X que hubiera en la posición indicada; devuelve lo que había antes, es decir X.
- `int size()` : devuelve el número de elementos en la lista.
- `Object[] toArray()` : devuelve un array cargado con los elementos de la lista.

# Implementación colecciones: Java

- Con la interface **Map**:
  - class **HashMap**
    - Es una implementación muy eficiente en cuanto a uso de memoria.
    - Es rápida en todas las operaciones.
    - Puede decirse que es un “array asociativo” de tamaño dinámico.
  - class **LinkedHashMap**
    - Es una implementación basada en listas encadenadas.
    - Respeta el orden de inserción, a cambio de ser más lenta.
  - class **TreeMap**
    - Es una implementación que garantiza el orden de las claves cuanto se itera sobre ellas.
    - Es más voluminosa y lenta.
  - class **Hashtable**
    - Similar a “HashMap” pero con métodos sincronizados, lo que permite ser usada en programas concurrentes.
    - Todo es más lento que con una HashMap.

# Métodos

- `void clear()`: elimina todas las claves y valores.
- `boolean containsKey(Object clave)`: devuelve true si alguna clave es “equals” a la indicada.
- `boolean containsValue(Object valor)` :devuelve true si algún valor es “equals” al indicado.
- `boolean equals(Object x)`: devuelve true si contiene las mismas claves y valores asociados.
- `V get(Object clave)`: devuelve el valor asociado a la clave indicada.
- `boolean isEmpty()`: devuelve true si no hay claves ni valores.

# Métodos II

- `Set<K> keySet()` : devuelve el conjunto de claves.
- `V put(K clave, V valor)` : asocia el valor a la clave; devuelve el valor que estaba asociado anteriormente, o `NULL` si no había nada para esa clave.
- `V remove(Object clave)`: elimina la clave y el valor asociado; devuelve el valor que estaba asociado anteriormente, o `NULL` si no había nada para esa clave.
- `int size()` número de asociaciones { clave, valor }
- `Collection<V> values()` devuelve una estructura de datos iterable sobre los valores asocia

# Implementación colecciones: Java

- Con la interface **Set**:
  - class **HashSet**
    - Es una implementación muy eficiente en cuanto a uso de memoria.
    - Es rápida en todas las operaciones.
  - class **TreeSet**
    - Es una implementación más lenta y pesada; pero presenta la ventaja de que el iterador recorre los elementos del conjunto en orden.



# Métodos

- `boolean add(E elemento)`: añade un elemento al conjunto, si no estaba ya; devuelve `true` si el conjunto crece.
- `void clear()` : vacía el conjunto.
- `boolean contains(E elemento)` : devuelve `true` si existe en el conjunto algún elemento “equals” al indicado.
- `boolean equals(Object x)`: devuelve `true` si uno y otro conjunto contienen el mismo número de elementos y los de un conjunto son respectivamente “equals” los del otro.
- `boolean isEmpty()`: devuelve `true` si el conjunto está vacío.
- `Iterator <E> iterator()`: devuelve un iterador sobre los elementos del conjunto.
- `boolean remove(E elemento)`: si existe un elemento “equals” al indicado, se elimina;  
devuelve `true` si varía el tamaño del conjunto.
- `int size()`: número de elementos en el conjunto

# Colecciones Jython

- A parte de poder utilizar las colecciones de Java **Jython** añade una serie de **colecciones de alto rendimiento**.
- Se encuentran dentro de **collections**:
  - deque
  - defaultdict
  - namedtuple
- `from collections import deque, defaultdict, namedtuple`

# deque

- `collections.deque([iterable[, maxlen]])`
  - Crea un nuevo objeto deque inicializado de izquierda a derecha con el iterable.
  - Es una generalización de las pilas y las colas.
  - Es compatible con subprocessos seguros y eficientes.
  - Se puede limitar la longitud con maxlen si no se indica es sin límite.
  - Se pueden añadir elementos por la izquierda y por la derecha.

# Métodos

- `append(i)`
- `appendleft(i)`
- `extend(iterable)`
- `extendleft(iterable)`
- `pop()`
- `popleft()`
- `rotate(n)`
  - Rota el deque n veces a la derecha, (si es negativo a la izquierda).
- Permite índices negativos, operador `in`, método: `copy` y `deepcopy`

# Ejemplos

```
from collections import deque
```

**# Solo deja los 10 últimos!**

```
d1 = deque([i for i in range(20)], 10)
```

```
print(d1)
```

```
d1.rotate(2)
```

```
print(d1)
```

```
deque([10, 11, 12, 13, 14, 15, 16, 17, 18, 19], maxlen=10)
deque([18, 19, 10, 11, 12, 13, 14, 15, 16, 17], maxlen=10)
Presione una tecla para continuar . . .
```

# defaultdict

- Devuelve un nuevo objeto similar a un diccionario.
- Es una **subclase** de **dict**.
- Al construir el objeto se puede indicar un **parámetro “default\_factory”** por defecto es **None**.
- Se puede indicar list, int, set, etc. Este tipo indicado condiciona el comportamiento del diccionario.

# Ejemplo

```
>>> s = [('yellow', 1), ('blue', 2), ('yellow', 3), ('blue', 4), ('red', 1)]
>>> d = defaultdict(list)
>>> for k, v in s:
...     d[k].append(v)
...
>>> d.items()
[('blue', [2, 4]), ('red', [1]), ('yellow', [1, 3])]
```

- Utiliza una lista para almacenar los valores de cada clave.
- Con la primera clave se crea una lista vacía, y se van añadiendo los valores que coinciden con la clave

# Ejemplo

- Si utilizamos un **int** como **default\_factory**.
- Podemos crear un histograma de una forma sencilla.

```
>>> s = 'mississippi'
>>> d = defaultdict(int)
>>> for k in s:
...     d[k] += 1
...
>>> d.items()
[('i', 4), ('p', 2), ('s', 4), ('m', 1)]
```



# namedtuple

- Asignan un significado a cada posición de la tupla y permiten un código de auto-documentación.
- Se pueden utilizar donde se usen las tuplas normales.
- Podemos acceder a las posiciones mediante el nombre.
- Es una subclase de tupla

# namedtuple

- `namedtuple(typename, field_names [, verbose])`
  - Se indica el nombre de la clase.
  - La lista de campos, puede ser una lista de cadenas (serán los campos de cada posición), se pueden separar por comas o por espacios en blanco.
  - Si `verbose` es `True` la definición de la clase se imprime justo antes de
  - El método `__repr__()` enumera los contenidos en pares de clave = valor

# Ejemplos

```
from collections import namedtuple
```

**# Con verbose = True imprime el contenido de la clase creada:**

```
Point = namedtuple('Point', 'x y', verbose=True)
```

```
p1 = Point(8, y = 99)
```

**# Indexación por posición o por nombre.**

```
print(p1[0], p1.y)
```

**# Permiten desempaquetado:**

```
x,y = p1
```

```
print(x,y)
```

# Ejemplos

- También se pueden crear objetos utilizando el método de clase `_make`.
- Pasando una lista o una tupla con los elementos:
  - `L = [11,22]`
  - `p2 = Point._make(L)`
  - `print(p2)`
  
  - `t = (8,9)`
  - `p3 = Point._make(t)`
  - `print(p3)`

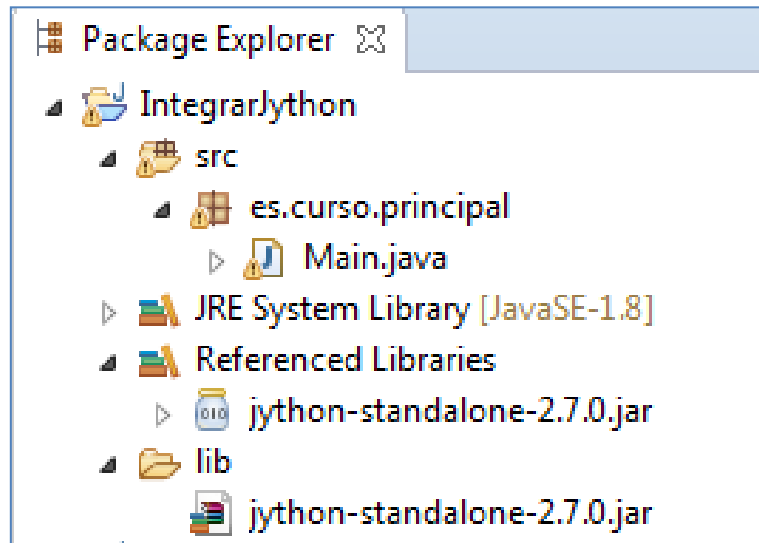
# Ejemplos

- Se puede obtener un diccionario ordenado con:
  - `mi_point._asdict()`
- Nuevas instancias reemplazando valores de los campos:
  - `>>> p = Point(x=11, y=22)`
  - `>>> p._replace(x=33)`
  - `Point(x=33, y=22)`

# PythonInterpreter

- Es un contenedor estándar con un **intérprete de Jython** para incrustarlo **en una aplicación Java**.
- Para integrar Jython en aplicaciones Java tenemos que añadir el Jar (***a la aplicación Java***)
  - **jython-standalone-2.7.0.jar**
- **JavaDoc:**
  - <http://www.jython.org/javadoc/org/python/util/PythonInterpreter.html>

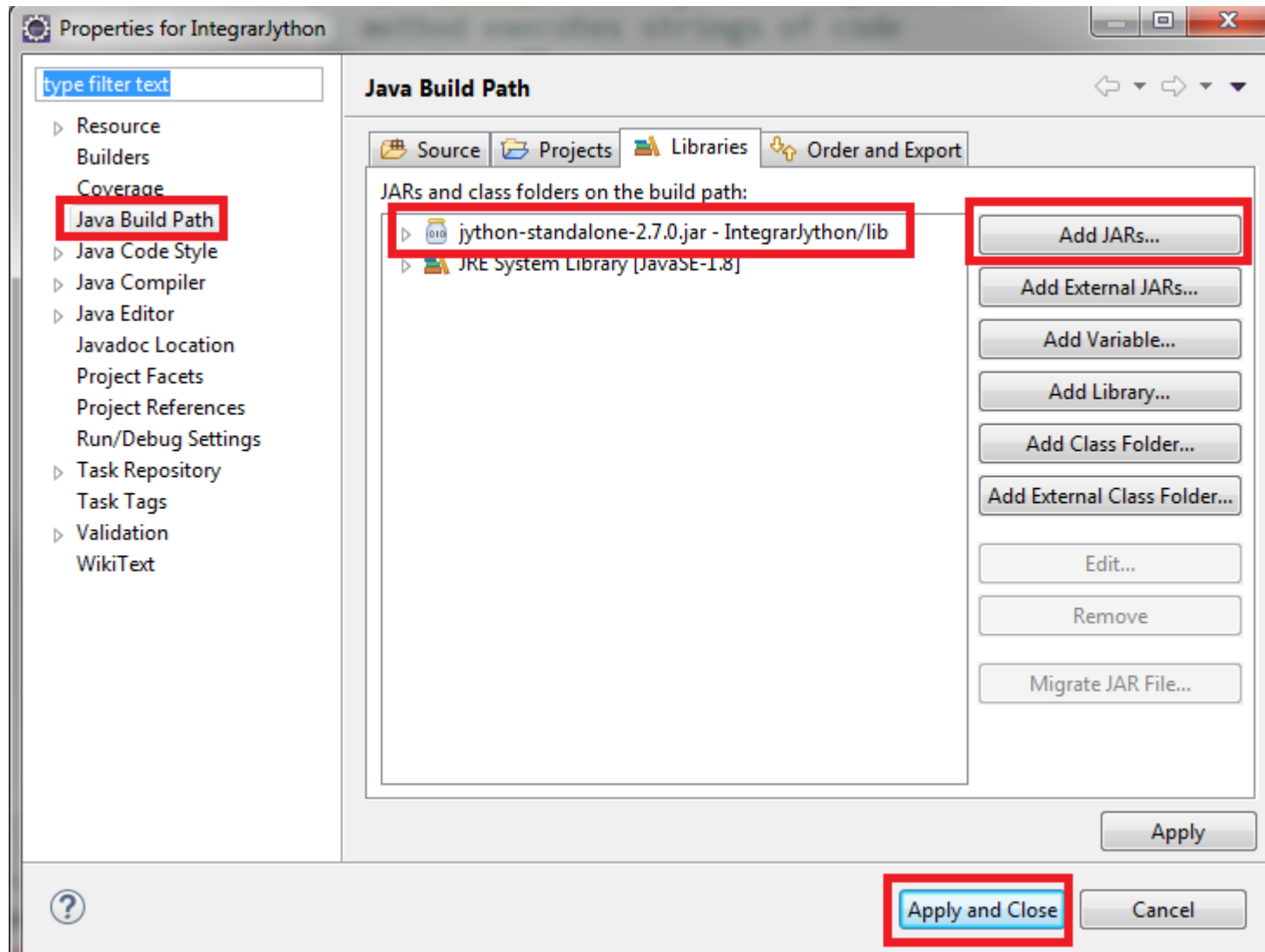
# Interactuar con el intérprete desde Java



- Crear un proyecto Java.
- Hay que **añadir** el **jar**, se copia a una carpeta **lib** dentro del proyecto:
- **jython-standalone-2.7.0.jar**

Para **vincular** el **jar** al **proyecto Java**, botón derecho sobre el proyecto:  
**Buildpath** → **Configure BuildPath**

# Configurar path al jar





# Ejemplo

```
import org.python.core.PyException;
import org.python.core.PyInteger;
import org.python.core.PyObject;
import org.python.util.PythonInterpreter;

public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws PyException {

        // Create an instance of the PythonInterpreter
        PythonInterpreter interp = new PythonInterpreter();

        // The exec() method executes strings of code
        interp.exec("import sys");
        interp.exec("print sys");

        // Set variable values within the PythonInterpreter instance
        interp.set("a", new PyInteger(42));
        interp.exec("print a");
        interp.exec("x = 2+2");

        // Obtain the value of an object from the PythonInterpreter and store it
        // into a PyObject.
        PyObject x = interp.get("x");
        System.out.println("x: " + x);
    }
}
```

# Enlaces

- **Librería estándar** de Python con Jython:
  - <http://www.jython.org/docs/library/indexprogress.html>
- Uso de **PythonInterpreter**:
  - <http://www.jython.org/jythonbook/en/1.0/JythonAndJavaIntegration.html#utilizing-pythoninterpreter>