Testing Jython scripts – FDMEE

Ejecutando scripts Jython de FDMEE en IDE's

Hola a todos, me encontré con un nuevo desafío dentro de mi ámbito laboral de la consultoría EPM y decidí investigar para encontrar la mejor solución.

Opté por contar mi experiencia en este desafío, para luego pasar a mostrarles la solución que personalmente me ayudó a agilizar mi desarrollo y espero transmitirles a mis pares una nueva práctica que les pueda resultar de ayuda.

El desafío era desarrollar integraciones en **FDMEE** (On-premise) utilizando gran cantidad de scripting en el lenguaje **Jython**, el cual, es uno de los lenguajes permitidos para utilizar en la aplicación.

Puede que no sea una tecnología reciente, pero muchas empresas hoy en día continúan utilizando esta herramienta en gran escala.

Experiencia

Los consultores que ya han desarrollado scripts en FDMEE conocen que la única manera de depurar/debuggear nuestro código es realizar constantemente ejecuciones de reglas de carga que ejecuten nuestros scripts, o realizar ejecuciones en el módulo de "**Script Execution**", el cual nos permite ejecutar scripts custom sin necesidad de realizar una ejecución de regla de carga. Esto, para luego monitorear el log generado por FDMEE y ver si este nos arrojó un error o si todo funciona bien.

Por cierto, el editor de código que nos brinda FDMEE no es muy confortable, ya que no posee una plugin **IntelliSense** de autocompletado, ni tampoco un plugin para colorear nuestro código.

La manera anteriormente nombrada para la depuración de nuestro código, puede ser algo tolerante cuando debemos desarrollar pequeños scripts que realicen una lógica simple, por ejemplo, trabajar con conexiones a bases de datos para extracción de información, crear archivos, cargar tablas de distintas bases de datos. Pero a medida que nuestro código se extiende cada vez más, nosotros como desarrolladores nos podemos sentir cegados ante la aparición de nuevos errores, excepciones, los cuales debemos depurar mediante ejecuciones continuas y luego con la lectura de los logs.

Este proceso de depuración de scripts a gran escala puede no ser la forma más rápida y eficaz para monitorear el código. Esto conlleva a lentificar el desarrollo de nuestros scripts y también y muy importante, no terminar de utilizando "clean code" (código limpio con buenas prácticas) ya que puede que cuando encontremos la solución al problema, no realicemos una limpieza del código, quitando las líneas o importaciones de librerías que no utilizamos.

Mi desafío de tener que desarrollar scripts con lógica gran cantidad de lógica, me llevó a investigar si existía alguna manera diferente de realizar mis depuraciones y ser más eficiente al escribir mi código.

Sebastian Tofano | CONTACTO: SEBASTIANTOFANO@GMAIL.COM
CONSULTOR EPM

Investigué dentro de la documentación oficial de **Oracle** y descubrí que existía una manera de testear mis scripts utilizando entornos de desarrollo. El entorno de desarrollo recomendado por Oracle es **Eclipse**.

¿Qué es un entorno de desarrollo?

En simples palabras es una aplicación, la cual nos proporciona todas las herramientas necesarias para un desarrollador. Entre la funcionalidad más simple que nos brinda, está la corrección gramatical del lenguaje utilizado, también, ver nuestro código coloreado en función de las sentencias que escribimos.

Si nos preguntamos para que nos podría servir tener un entorno de desarrollo, podríamos darnos cuenta que lo nombrado anteriormente ya es muy útil, pero, además, Eclipse nos va a permitir configurar localmente un intérprete **Python** (o **Jython**), que se utiliza para la ejecución de scripts en FDMEE.

La documentación oficial es la siguiente, https://docs.oracle.com/cd/E57185_01/FDMAG/ch08s08s04.html, aunque también existe documentación no oficial donde se aplican los pasos a seguir para configurar Eclipse y conectarse a FDMEE en modo desarrollo.

Al descubrir esta documentación, investigué si esto podría ayudarme a depurar mi código más rápido y efectivamente me ayudó a poder testear mis scripts localmente con Eclipse y tener una consola donde poder ver los resultados de la ejecución, en comparación con FDMEE donde debería revisar un log. Además, logré detectar más fácilmente errores y excepciones las cuáles no estaba teniendo en cuenta.

Para contarles en simples palabras, estoy utilizando Eclipse para correr mis scripts localmente antes de implementarlos en FDMEE. De esta manera, llevo un control de la lógica implementada, ya sea conexiones a bases de datos, creación de archivos en diferentes directorios locales como propios de la aplicación, cargas de información dentro de bases de datos, peticiones **REST** a ambientes **EPM Cloud**, entre otras.

Tal vez sea un poco confuso entender por qué localmente puedo ejecutar scripts que utilicen librerías que solamente existen en el ambiente de FDMEE, como, por ejemplo, la librería principal de FDMEE llamada **fdmAPI**. A esta altura si hemos leído el enlace oficial de Oracle, podemos saber que esta librería se puede descargar a nuestro equipo e importarlas en nuestro proyecto donde desarrollaremos y testearemos nuestros scripts.

De todos modos**, en este documento vamos a mostrar un enfoque distinto al que nos plantea la documentación Oficial**. Lo que vamos a buscar es desarrollar nuestras propias clases o funciones que nos permitan detectar en que entorno se está ejecutando nuestro script, auto-detectar si la ejecución se lanzó sobre FDMEE o si se lanzó localmente desde nuestro equipo y a partir de esto, permita seleccionar las funciones correctas que se necesiten correr en el entorno.

Para poner un simple ejemplo, si estamos en el entorno de FDMEE, para mostrar un mensaje en el log para realizar una depuración podemos utilizar la función **fdmAPI.logInfo('mensaje')**

Mientras que, en nuestro entorno local, para poder visualizar el mensaje en la consola de Eclipse necesitamos que la función anterior sea reemplazada por un simple **print('mensaje')**

Otro ejemplo, puede ser el manejo de directorios de los diferentes entornos, ya que si ejecutamos el script para crear o leer un archivo en el entorno de FDMEE necesitamos que el directorio se base en el árbol de carpetas del servidor de FDMEE (**fdmContext['OUTBOXDIR']** por ejemplo).

Mientras que, en nuestro entorno local, si queremos leer o crear un archivo vamos a necesitar que este directorio sea el directorio local de nuestro equipo.

Para finalizar este aporte, voy a dejar un manual de configuración para desarrolladores que les interese utilizar este proceso en su depuración de scripts.

Espero que sea un aporte que pueda ayudar a la comunidad de consultores.

¡Muchas gracias por su atención!

A continuación el manual de configuración…

Manual de Configuración

Ya dándole un enfoque más técnico a este contenido, vamos a mostrar los pasos necesarios para configurar en nuestro equipo este entorno de desarrollo.

Hay pasos en los que necesitarán contar con conocimientos técnicos, como, por ejemplo, descargar aplicaciones, instalar módulos de Eclipse, librerías, drivers.

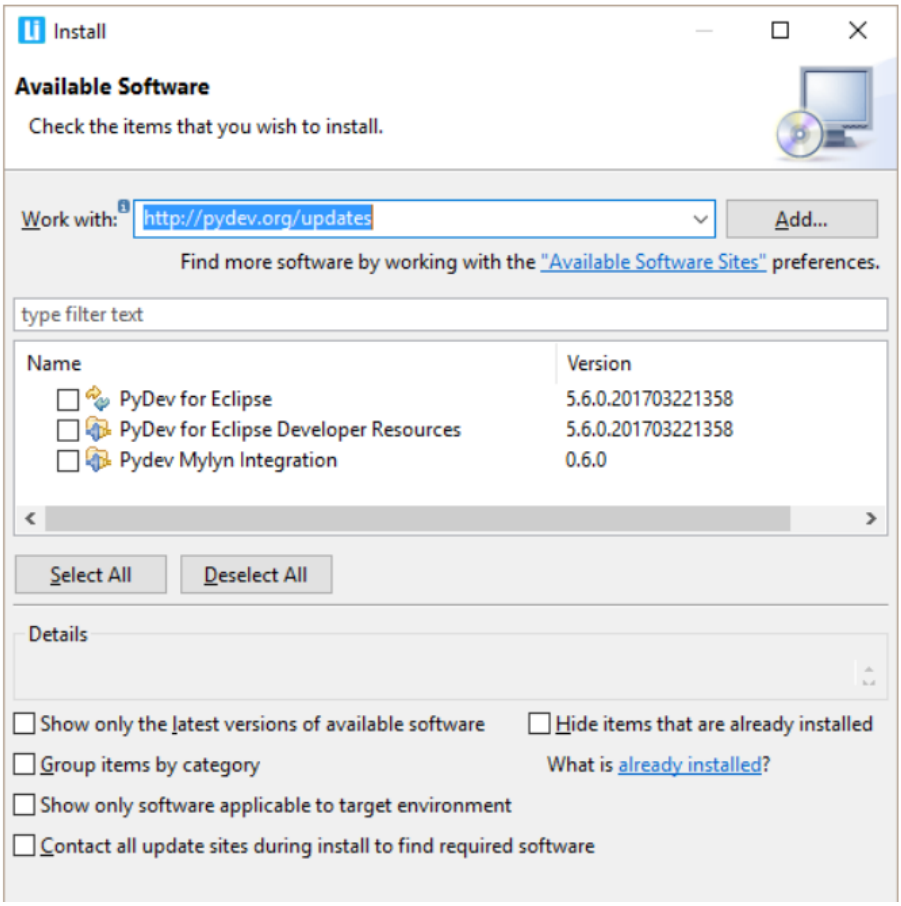
Como ya saben cada equipo es un mundo diferente, entonces tal vez puedan enfrentarse con distintos stoppers en la configuración.

Pasos:

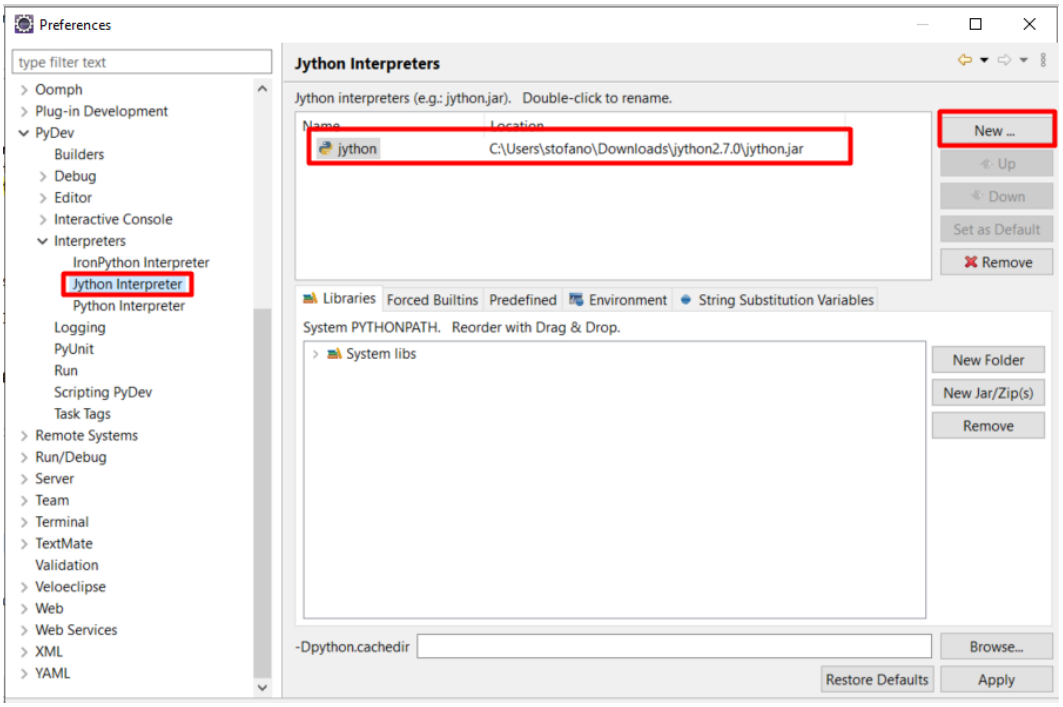
1. Contar con el **SDK** de **Java** instalado en nuestro equipo. (JRE, JDK)
   Recomendamos descargarse cualquier reléase de la **Java 8** (versión 1.8)
   https://www.java.com/es/download/

2. Instalar el entorno de desarrollo **Eclipse**, recomendado por Oracle:
   https://www.eclipse.org/eclipseide/

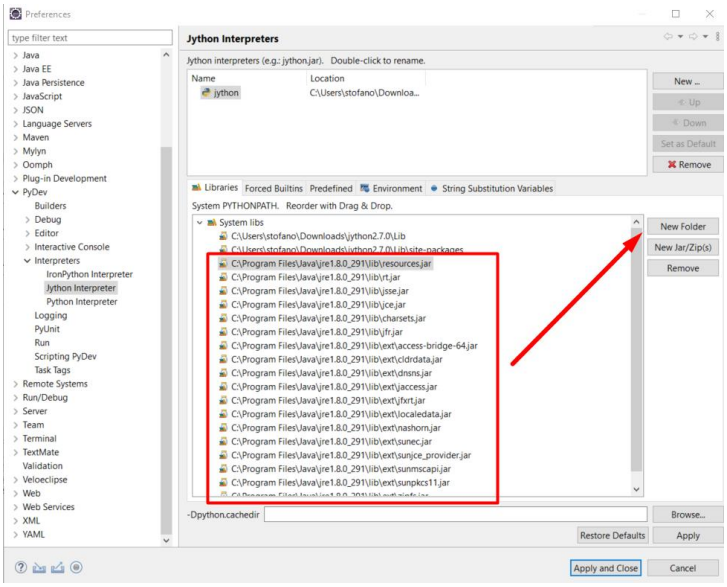
3. Instalar **PyDev** dentro de Eclipse. Es recomendable seguir las instrucciones en la web oficial
   https://www.pydev.org/manual_101_install.html



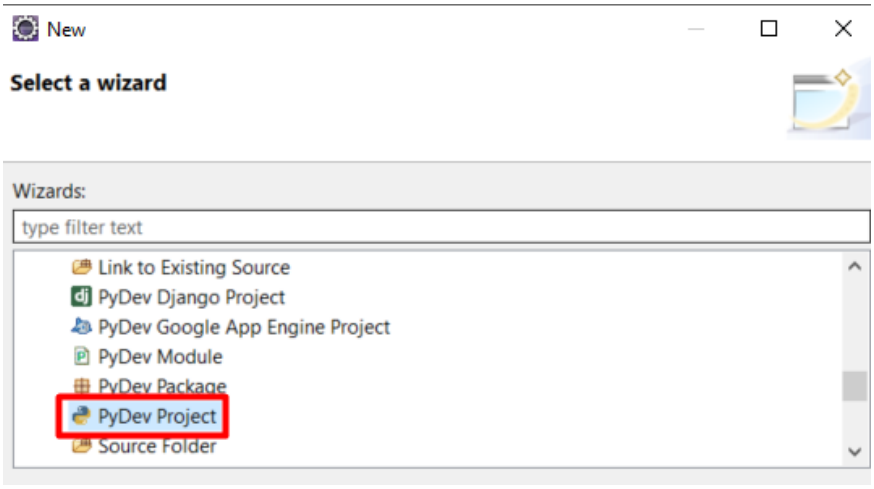
4. Ahora que tenemos **PyDev** instalado, debemos configurar el intérprete de **Jython** sobre el cuál se ejecutarán nuestros scripts.
   En este paso debemos descargar la librería de **Jython** desde la página oficial, puede ser la versión **2.5** o **2.7**.
   La versión que utiliza **FDMEE** es la **2.5**, por lo que lo recomendable pueda ser descargar esta versión.
   En este manual, se descargará la versión **2.7** para poder testear funciones nuevas de esta versión que no funcionan de manera por defecto en **FDMEE**.
   https://www.jython.org/download
   Esta librería será la que tendremos que configurar añadir a la preferencia del intérprete de **Jython**.

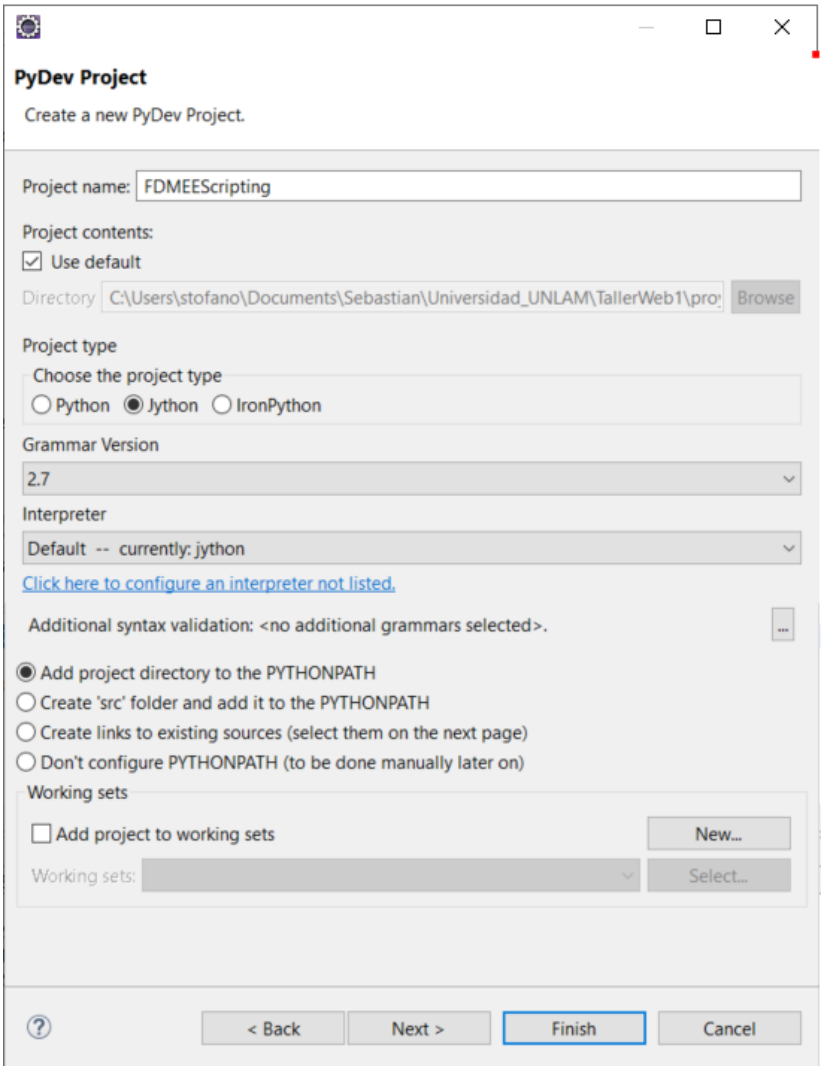


5. Además, nos aseguraremos de tener importadas nuestras **librerías de Java**. De no tenerlas, debemos incluirlas.

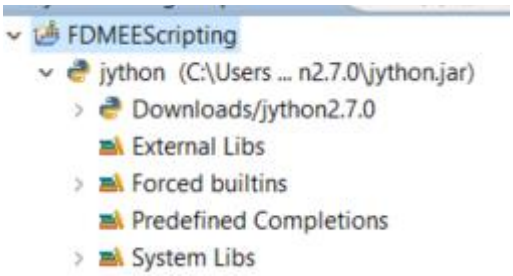


6. Ahora ya podemos iniciar un proyecto Python dentro de Eclipse. (Aunque lo ejecutaremos con el intérprete de Jython).
   Aquí podremos configurar que tipo de proyecto vamos a crear, seleccionaremos "**Jython**", y luego podremos elegir diferentes versiones de gramática para nuestro código, si nuestro interprete es de la versión 2.7, nos convendrá elegir la versión 2.7 de gramática.



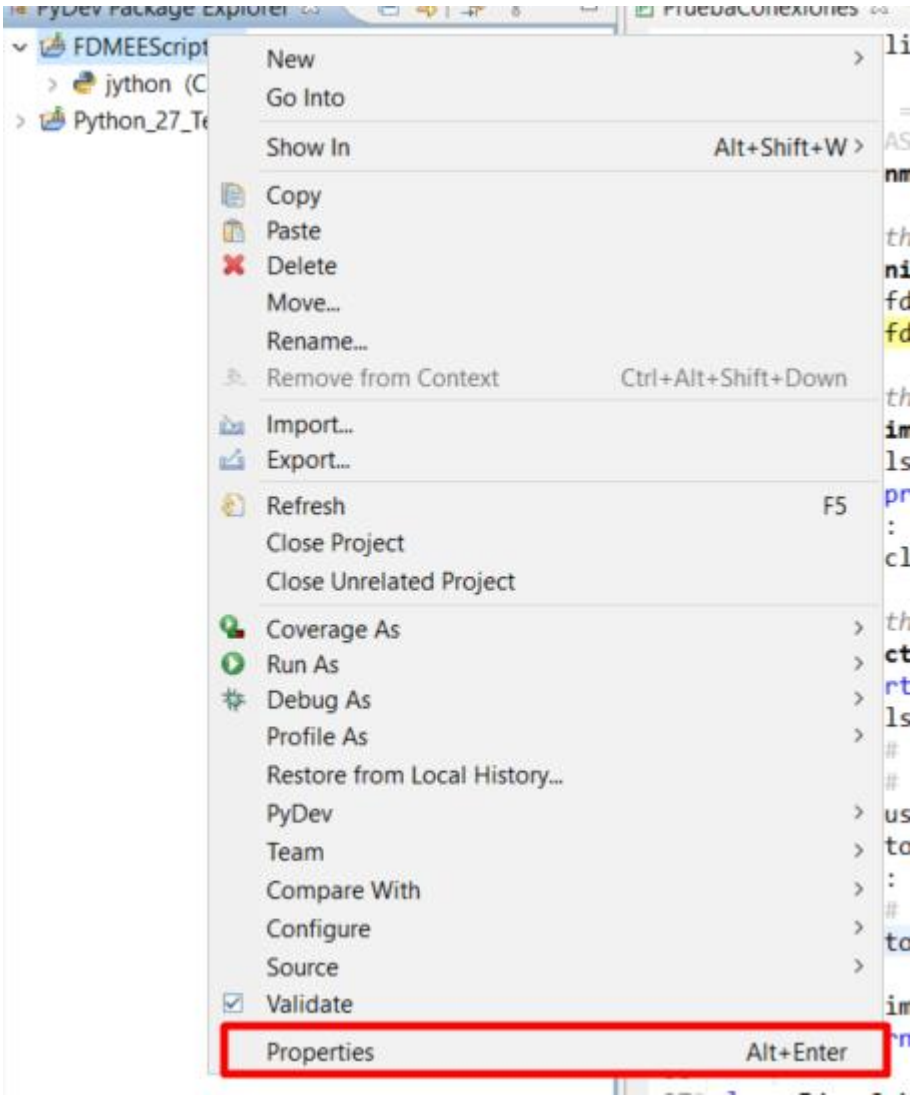


7. Una vez creado nuestro proyecto,

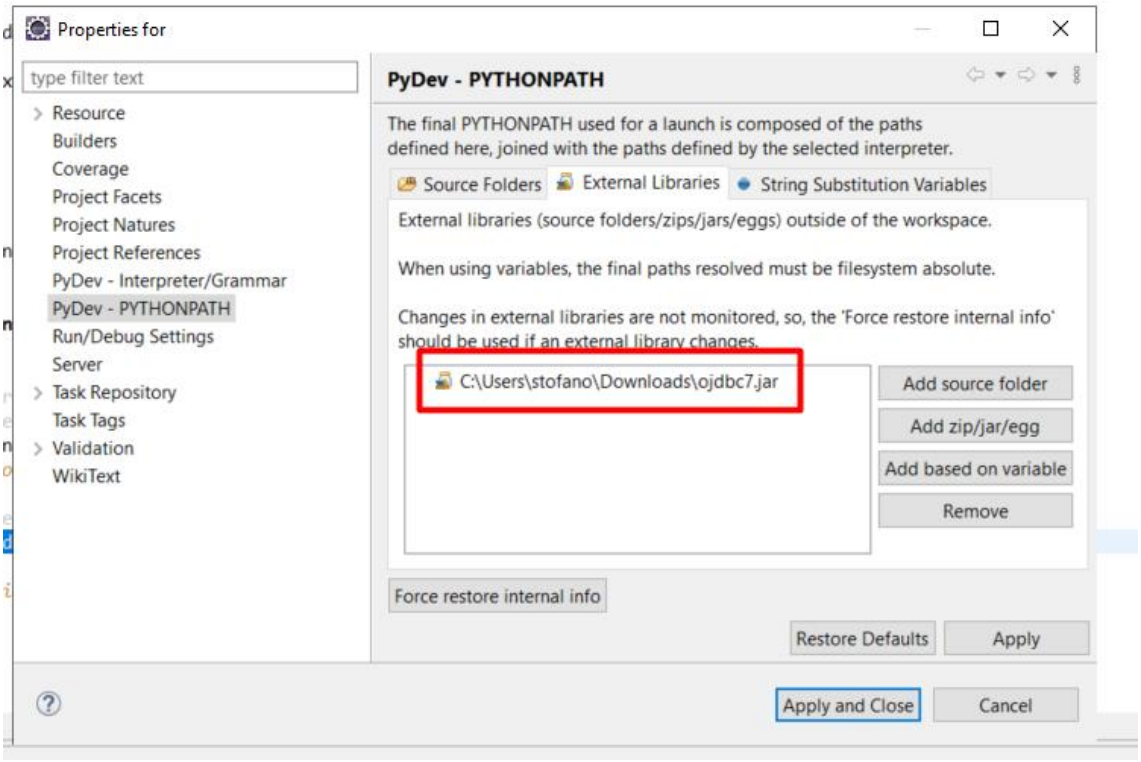


Lo recomendable es **importar librerías/drivers** de las bases de datos que utilizaremos en nuestros scripts.

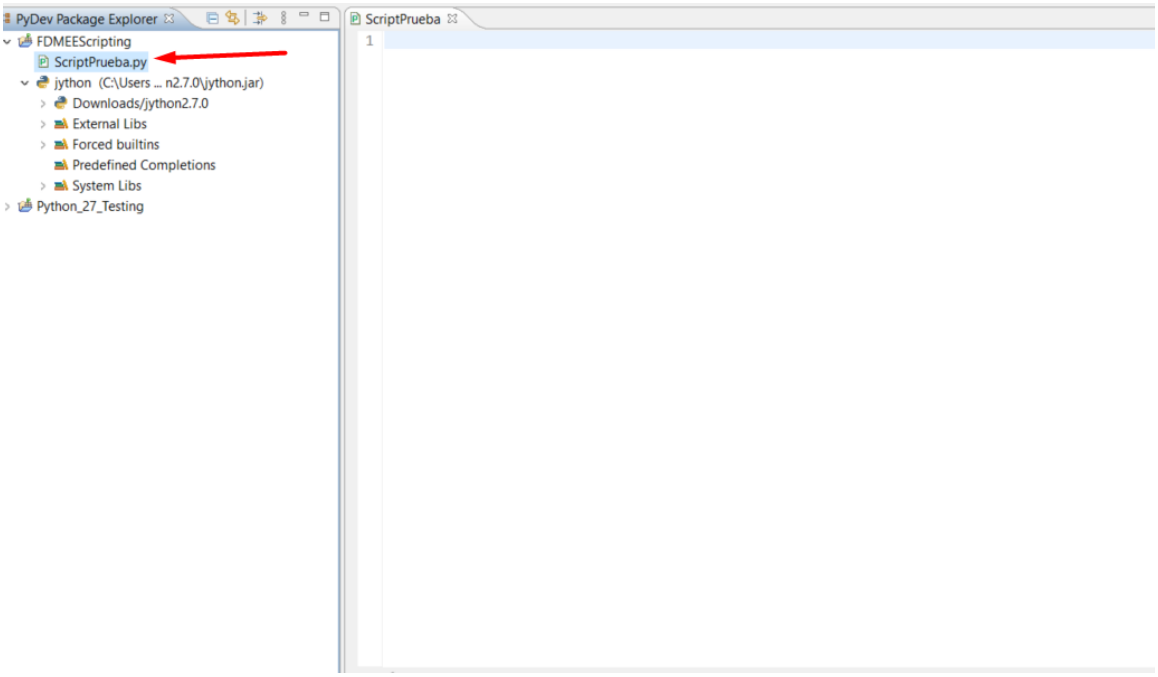
Iremos a las propiedades de nuestro proyecto



E incluiremos el driver o librería que necesitemos para la conexión a nuestras bases de datos.



8. Crearemos un nuevo archivo con extensión Python, el cual será la base para escribir nuestro script a testear.



9. **<u>Aquí llega una de las partes más interesantes</u>**, en nuestro script vamos a crear las clases por defecto, que nos permitirán detectar en que entorno estamos ejecutando el script, si en FDMEE o localmente.
   Esto nos servirá para que el script pueda ser multiplataforma y poder probarlo tanto localmente como en FDMEE sin tener que realizarle modificaciones.

La clase **FdmeeOrLocal**, nos permitirá detectar automáticamente el entorno de ejecución

```
# coding=UTF-8

# INICIO: CLASE PARA INICIALIZAR CONTEXTO LOCAL O FDMEE
class FdmeeOrLocal:

    @classmethod
    def get_entorno_FDMEE_or_LOCAL(cls):
        import sys
        ver = sys.platform.lower()
        if ver.startswith('java'):
            import java.lang
            ver = java.lang.System.getProperty("os.name").lower()
        entorno = 'LOCAL' if ver.find('win') != -1 else 'FDMEE'
        return entorno
```

La clase **Environment**, nos permitirá definir qué funciones deben utilizarse para un entorno y cuales otras deben utilizarse para el otro entorno.
En nuestro equipo debemos contar con la variable de entorno "**HOMEPATH**" configurada

```
class Environment:

    @classmethod
    def definirEnvironment(cls, fdmAPI, fdmContext):
        cls.fdmAPI = fdmAPI
        cls.fdmContext = fdmContext

    @classmethod
    def imprimir(cls, mensaje):
        if cls.fdmAPI is None:
            print(mensaje)
        else:
            cls.fdmAPI.logInfo(mensaje)

    @classmethod
    def directorio_segun_enviroment(cls, directorio = ''):
        import os
        if cls.fdmAPI is None:
            user_home = os.environ['HOMEPATH']
            # Ingrese aqui su directorio local
            total_dir = '%s/Downloads/' % (user_home)
        else:
            # Ingrese aqui el directorio de FDMEE que va a utilizar
            total_dir = str(cls.fdmContext['OUTBOXDIR'].replace('\\','/')) + '/' + directorio + '/'

        cls.imprimir('\tLa ubicacion seleccionada es: ' + total_dir)
        return total_dir
```

10. Una vez creadas estas clases podremos ejecutar una lógica para reconocer e instanciar/inicializar el entorno.


```
47 # INICIO: INICIALIZACION DE ENVIRONMENT LOCAL O FDMEE
48 if FdmeeOrLocal.get_entorno_FDMEE_or_LOCAL()  == 'LOCAL':
49     fdmAPI = None
50     fdmContext = None
51
52 Environment.definirEnvironment(fdmAPI, fdmContext)
53 # FIN: INICIALIZACION DE ENVIRONMENT LOCAL O FDMEE
```

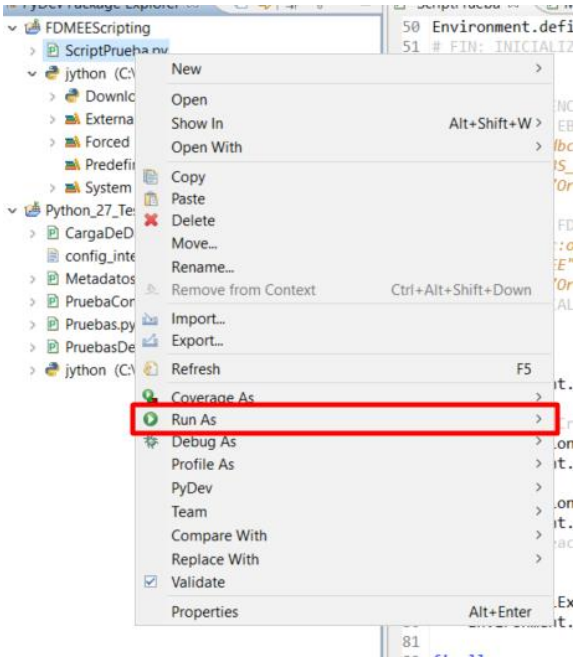
11. De aquí en adelante ya podremos desarrollar nuestra lógica del script.
    IMPORTANTE: Al desarrollar debemos tener en cuenta que para utilizar funciones que dependan del entorno de ejecución, debemos utilizar la función única creada en la clase "**Environment**".
    Por ejemplo, si deseamos conectarnos a bases de datos e imprimir por pantalla si ha ocurrido exitosamente, debemos utilizar la función única "**imprimir**" de la clase "**Environment**", que permite reconocer el entorno de ejecución y ejecutar la función correspondiente.

    Ejemplo de lógica de conexión a bases de datos:

```
84  try:
85      Environment.imprimir("Paso 1: Establecer conexiones a bases de datos")
86
87      # INICIO: Creacion de conexiones
88      ebs_conexion = sql.DriverManager.getConnection(jdbc_EBS, user_EBS, password_EBS)
89      Environment.imprimir("\tConexion exitosa a la base de datos de: EBS")
90
91      fdm_conexion = sql.DriverManager.getConnection(jdbc_FDM, user_FDM, password_FDM)
92      Environment.imprimir("\tConexion exitosa a la base de datos de: FDMEE")
93      # FIN: Creacion de conexiones
94
95
96  except sql.SQLException, e:
97      Environment.imprimir("Se ha producido el siguiente error: " + str(e))
98
99  finally:
100     ebs_conexion.close()
101     fdm_conexion.close()
```

    De necesitar progresivamente utilizar diferentes funciones para diferentes entornos, va a ser necesario modificar la clase "Environment" de acuerdo a nuestra necesidad. Creando un nuevo método de clase.

12. Para ejecutar nuestro script debemos hacerlo con el intérprete de Jython -> **"Jython Run"**



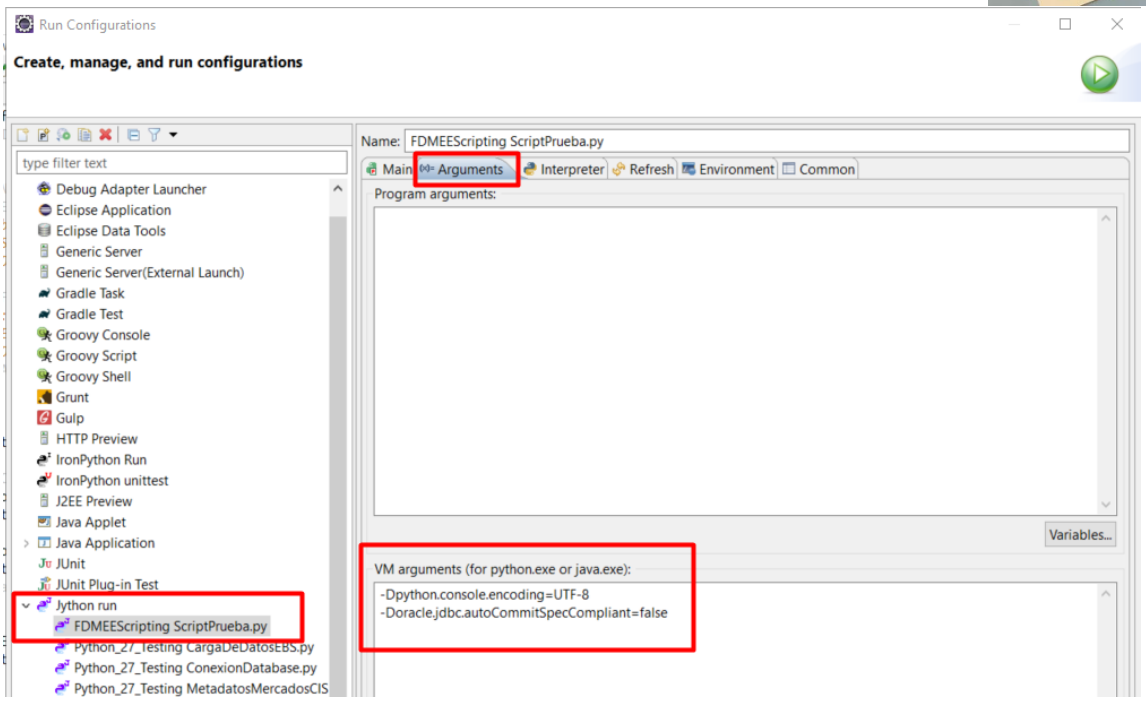
13. Veremos en la consola de trabajos de Eclipse, como va ejecutando nuestro código en tiempo de ejecución a medida que es interpretado.

```
66
67 try:
68     Environment.imprimir("Paso 1: Establecer conexiones a bases de datos")
69
70     # INICIO: Creacion de conexiones
71     ebs_conexion = sql.DriverManager.getConnection(jdbc_EBS, user_EBS, password_EBS)
72     Environment.imprimir("\tConexion exitosa a la base de datos de: EBS")
73
74     fdm_conexion = sql.DriverManager.getConnection(jdbc_FDM, user_FDM, password_FDM)
75     Environment.imprimir("\tConexion exitosa a la base de datos de: FDMEE")
76     # FIN: Creacion de conexiones
77
78
79 except sql.SQLException, e:
80     Environment.imprimir("Se ha producido el siguiente error: " + str(e))
81
82 finally:
83     ebs_conexion.close()
84     fdm_conexion.close()
```

```
Console  PyUnit
<terminated> ScriptPrueba.py [C:\Program Files\Java\jre1.8.0_291\bin\javaw.exe]
console: Failed to install '': java.nio.charset.UnsupportedCharsetException: cp0.
Paso 1: Establecer conexiones a bases de datos
        Conexion exitosa a la base de datos de: EBS
        Conexion exitosa a la base de datos de: FDMEE
```

14. Si quisieran quitar el error que les figura en la primera línea de la consola, deberán incluir el siguiente comando como argumento de la **VM**.
    -Dpython.console.encoding=UTF-8

    Además, incluiremos también el siguiente comando para solucionar problemas con el Auto-Commit en las bases de datos Oracle.
    -Doracle.jdbc.autoCommitSpecCompliant=false



NOTA: En breve subiré las clases custom en un repositorio de GitHub de modo que cualquier consultor pueda colaborar en el proyecto.
https://github.com/sebastiantofano

Código de ejemplo:

```
# coding=UTF-8
import java.sql as sql

# INICIO: CLASE PARA INICIALIZAR CONTEXTO LOCAL O FDMEE
class FdmeeOrLocal:

    @classmethod
    def get_entorno_FDMEE_or_LOCAL(cls):
        import sys
        ver = sys.platform.lower()
        if ver.startswith('java'):
            import java.lang
            ver = java.lang.System.getProperty("os.name").lower()
        entorno = 'LOCAL' if ver.find('win') != -1 else 'FDMEE'
        return entorno

class Environment:
    @classmethod
    def definirEnvironment(cls, fdmAPI, fdmContext):
        cls.fdmAPI = fdmAPI
        cls.fdmContext = fdmContext

    @classmethod
    def imprimir(cls, mensaje):
        if cls.fdmAPI is None:
            print(mensaje)
        else:
            cls.fdmAPI.logInfo(mensaje)

    @classmethod
    def directorio_segun_enviroment(cls, directorio = ''):
        import os
        if cls.fdmAPI is None:
            # Ingrese aqui su directorio local
            user_home = os.environ['HOMEPATH']
            total_dir = '%s/Downloads/' % (user_home)
        else:
            # Ingrese aqui el directorio de FDMEE que va a utilizar
            total_dir = str(cls.fdmContext['OUTBOXDIR'].replace('\\','/')) + '/' + directorio
+ '/'

        cls.imprimir('\tLa ubicacion seleccionada es: ' + total_dir)
        return total_dir
# FIN: CLASE PARA INICIALIZAR CONTEXTO LOCAL O FDMEE

# INICIO: INICIALIZACION DE ENVIRONMENT LOCAL O FDMEE
if FdmeeOrLocal.get_entorno_FDMEE_or_LOCAL()  == 'LOCAL':
    fdmAPI = None
    fdmContext = None

Environment.definirEnvironment(fdmAPI, fdmContext)
# FIN: INICIALIZACION DE ENVIRONMENT LOCAL O FDMEE


#INICIO: CREDENCIALES DE LAS BASES DE DATOS
#Credenciales EBS
jdbc_EBS = "jdbc:oracle:thin:@<server>:<port>/EBS"
user_EBS = "EBS_user"
password_EBS="Oracle2021"

#Credenciales FDMEE
jdbc_FDM="jdbc:oracle:thin:@<server>:<port>:EPM"
user_FDM="FDMEE"
password_FDM="Oracle2021"
```


```
#FIN: CREDENCIALES DE LAS BASES DE DATOS


try:
    Environment.imprimir("Paso 1: Establecer conexiones a bases de datos")

    # INICIO: Creacion de conexiones
    ebs_conexion = sql.DriverManager.getConnection(jdbc_EBS, user_EBS, password_EBS)
    Environment.imprimir("\tConexion exitosa a la base de datos de: EBS")

    fdm_conexion = sql.DriverManager.getConnection(jdbc_FDM, user_FDM, password_FDM)
    Environment.imprimir("\tConexion exitosa a la base de datos de: FDMEE")
    # FIN: Creacion de conexiones


except sql.SQLException, e:
    Environment.imprimir("Se ha producido el siguiente error: " + str(e))

finally:
    ebs_conexion.close()
    fdm_conexion.close()
```