Mémo : Principes et exemples de Jython dans Openrefine

Auteur : Mathieu Saby

Licence CC-BY

Historique

|  |  |  |
| --- | --- | --- |
| V1.1 | 1/11/2020 | Exemples |
| V1.0 | 12/11/2018 | Création, à compléter |

Les transformations et facettes personnalisées sont basées sur des « expressions », équivalent des formules dans Excel. Les expressions peuvent être écrites en langage GREL (*Google Refine Expression Language*, ou *General Refine Expression Language*) spécifique d’Openrefine, en Jython (implémentation de Python en langage Java) ou en Clojure. GREL est utilisé plus souvent que Jython et Clojure, en général réservés à des cas complexes.

Ce document présente rapidement Jython 2.7.

Table des matières

[1. Principes de Jython 2](#_Toc529942856)

[2. Utiliser des librairies non standards 3](#_Toc529942857)

[2.1. Ajouter une librairie Java 4](#_Toc529942858)

[2.2. Ajouter une librairie Python 4](#_Toc529942859)

[3. Cas d’usage 5](#_Toc529942860)

# Principes de Jython

Présentation générale dans la documentation d’Openrefine: [Using Jython as your Expression](https://github.com/OpenRefine/OpenRefine/wiki/Jython)

Documentation officielle de Jython (pas à jour, basée sur Python 2.5) : [jython.org](https://jython.org)

Dans certains cas il peut être utile d’utiliser le langage Jython au lieu de GREL.

Jython est une version de Python dont l’implémentation repose sur le langage Java.

**Utilité** dans Openrefine: outrepasser les limites structurelles ou fonctionnelles de GREL (instructions chaînées, impossibilité de créer des fonctions, d’utiliser des variables intermédiaires…)

**Limitations :**

* L’interface de saisie des formules n’est pas adaptée à Python (besoin d’insérer manuellement les indentations du code ; pas d’autocomplétion…)
* Jython permet d’utiliser la version 2.7de Python mais pas la version 3.X.
* Les modules Python écrits en C ne sont pas utilisables (ex : Numpy)
* L’installation de nouveaux modules est possible mais assez complexe

**Particularité de l’usage de Jython dans Openrefine :**

* Les valeurs spéciales liées au modèle de données d’Openrefine sont directement utilisables (value, row, cell, row…)
* Les noms de colonne ou les champs d’objets peuvent être référencés avec la syntaxe ["nom"] ou .nom (s’il n’y a pas d’espace dans le nom)
* La valeur retournée doit être précédée de *return*
* Le type d’une valeur *null* est *None*

Ex : retourne la valeur de chaque cellule

return value

Ex : retourne la valeur de chaque cellule de la colonne nomcol

return row.cells.nomcol.value

Ex : retourne 0 pour chaque numéro de ligne paire, 1 pour chaque numéro impair

return row.index%2

Ex : si la cellule contient du texte, retourne les caractères [2 à avant-dernier], sinon retourne une erreur.

return value[1:-1]

Ex : retourne une valeur *null* (impossible en GREL)

return None

Ex : retourne une valeur *null* (impossible en GREL)

if value is not None:

 return "non null"

Ex: renvoie une information en fonction du contenu de 3 colonnes:

if cells.prix\_u.value\*cells.qte.value > 600 and cells.utile.value=="faible":

 return "trop cher"

else:

 return "ok"

Il est possible d’utiliser la **syntaxe standard** de Python :

* **définition et utilisation de variables**

a = 10

b = "texte"

c =

return doublex (value)

* **itération, boucles, conditions**

tmp = ""

liste = ["a","b","c"]

for i in range(10) :

 for j in liste:

 tmp=tmp+j

return tmp

* **conditions**

if value == "oui" or value == "yes" :

 return 1

elif value == "non" or value == "no" :

 return 0

else :

 return None

* **définition et utilisation de fonctions**

def doublex (x):

 return 2\*x

return doublex (value)

Les **fonctions** utilisables en standard: Cf <https://docs.python.org/2/library/>

Ex (expressions régulières) :

import re

m = re.search('(?<=abc)def', 'abcdef')

return m.group(0)

Ex (manipulation de fichiers)

import sys

m = re.search('(?<=abc)def', 'abcdef')

return m.group(0)

Ex (requête HTTP )

import urllib2

url = "http://text-processing.com/api/sentiment/"

data = "text=very bad feeling"

post = urllib2.urlopen(url, data)

return post.read()

Renvoie : {"probability": {"neg": 0.85254920138939672, "neutral": 0.067537718594657167, "pos": 0.14745079861060323}, "label": "neg"}

Divers :

Exemple :

Utiliser une API d’analyse de sentiments

<http://text-processing.com/>

Documentation :

<http://text-processing.com/docs/index.html>

Les requêtes doivent être passées en POST

# Utiliser des librairies non standards

Il est possible d’utiliser des libraires ne faisant pas partie de la librairie standard installée avec Jython : modules téléchargés ou créés par l’usager.

Après installation d’une librairie, Openrefine doit être fermé et redémarré.

Voir : [Extending-Jython-with-pypi-modules](https://github.com/OpenRefine/OpenRefine/wiki/Extending-Jython-with-pypi-modules)

Jython peut utiliser des librairies Python mais aussi des librairies Java.

Pour utiliser un fichier python2 (.py), déposez le dans le chemin.

## Ajouter une librairie Java

1. Télécharger le fichier .jar contenant la librairie
2. Déplacer le dans le dossier webapp/WEB-INF/lib d’Openrefine (sous Mac : /Applications/OpenRefine.app/Contents/Resources/webapp/WEB-INF)
3. Redémarrer Openrefine

Ex : librairie pour analyser des numéros de téléphone : jar téléchargeable sur <https://mvnrepository.com/artifact/com.googlecode.libphonenumber/libphonenumber/8.9.16>

Télécharger, déplacer dans le dossier et redémarrer

from com.google.i18n.phonenumbers import PhoneNumberUtil

from com.google.i18n.phonenumbers.PhoneNumberUtil import PhoneNumberFormat

phoneUtil = PhoneNumberUtil.getInstance()

number = phoneUtil.parse(value, 'FR')

formatted = phoneUtil.format(number, PhoneNumberFormat.NATIONAL)

valid = phoneUtil.isValidNumber(number)

if valid == 1:

 return formatted

Résultat



## Ajouter une librairie Python

* télécharger Jython 2.7 et installer (en mode standard) en dehors d’Openrefine. Les outils pip et setuptools seront installés automatiquement.
* (facultatif ?) ajouter le répertoire d’installation de Jython dans le PATH de son système d’exploitation (sous Windows, C:\jython2.7x\bin ; Mac, /Users/username/jython2.7.0/bin)
* ouvrir une console Windows ou un Terminal,
* installer les modules requis avec la commande jython -m pip install nom\_module

Ex pour le module <https://pypi.org/project/address-formatter/> qui analyse des adresses postales :

jython -m pip install address-formatter

Ex pour le module <https://pypi.org/project/Unidecode/> qui normalise des caractères unicode :

jython -m pip install unidecode

* Démarrer Openrefine
* Dans l’éditeur d’expressions, saisir une formule en commençant par ajouter au PATH le dossier site-package de Jython et importer le package ou le fichier .py à utiliser

Ex : parser des adresses postales

import sys

sys.path.append(r'C:\jython2.7.0\Lib\site-packages') [Windows]

sys.path.append('/Users/username/jython2.7.0/Lib/site-packages') [Mac]

from address import AddressParser, Address

ap = AddressParser()

return ap.parse\_address(value)

Ex : supprimer des caractères diacritiques avec la librairie unidecode

import sys

sys.path.append(r'C:\jython2.7.0\Lib\site-packages') [Windows]

sys.path.append(‘/Users/username/jython2.7.0/Lib/site-packages') [Mac]

from unidecode import unidecode

return unidecode(value)

# Cas d’usage

* Supprimer les diacritiques (sans installer de librairie supplémentaire)

import unicodedata

def strip\_accents(s):

 return ''.join(c for c in unicodedata.normalize('NFKD', s) if unicodedata.category(c) != 'Mn')

return strip\_accents(value)

* Renvoyer le plus long élément d’un tableau (source : https://stackoverflow.com/questions/51799543/how-to-get-the-longest-string-in-an-array-in-openrefine)

return max(['a','aaa','aaaa','aa'], key=len)

* Rechercher toutes les occurrences d’un motif (source : https://stackoverflow.com/questions/44009616/return-multiple-results-in-openrefine-using-python-jython-regex)

import re

g = re.findall("motif", value)

return g

* Concaténer deux tableaux

return [1,2,3] +[3,4,5]

* Parser des données en XML

from xml.etree import ElementTree

element = ElementTree.fromstring(value.encode("utf-8"))

listeResultats = element.findall(".//author[@country='GB']/..")

titreauteur = []

for resultat in listeResultats:

 titreauteur.append ( resultat.find('title').text + " / " + resultat.find('author').text )

return "|".join (titreauteur)

* Parser des données en JSON

import json

donnees = json.loads (value)

for cle, valeur in donnees.items():

 return (valeur['name'])

* Envoyer des données sur le web avec une requête POST (source : https://programminghistorian.org/en/lessons/fetch-and-parse-data-with-openrefine)

import urllib2

url = "http://text-processing.com/api/sentiment/"

data = "text=what is the sentiment of this sentence"

post = urllib2.urlopen(url, data)

return post.read()