Rapport de stage

Stephane Robin

30 mars 2020

Table des matières

Pour tester les codes du projet, nous avons créé un environnement de développement avec l'outil de création d'environnement virtuel *venv*. Ce dernier crée un dossier contenant tous les exécutables nécessaires pour utiliser les modules d'un projet Python. Nous avons défini Python3.6.5 comme version locale de travail dans cet environnement grâce au module *pyenv*. Nous avons également installé les dépendances dans cet environnement à l'aide de l'outil *pip3*. Notre environnement de développement contenait donc :

- pkg-resources==0.0.0 (installé automatiquement)
- xmltodict==0.12.0
  Le module xmltodict permet de lire du code XML comme si il s'agissait de code JSON. Il permet une lecture rapide des fichiers.s
- subprocess.run==0.0.8
  Le module subprocess permet de gérer de nouveaux processus, de se connecter à leurs flux d'input/output/erreurs. Il remplace plusieurs modules dépréciés : os. system, os.spawn*, os.popen*, popen2.*, commands.*
- openpyxl==3.0.3
  Le package openpyxl permet de lire et d'écrire dans des fichiers Excel de type xlsx, xlsm, xltx, xltm.
- et-xmlfile==1.0.1 (installé avec le module openpyxl)
- jdcal==1.4.1 (installé avec le module openpyxl)
- xlrd==1.2.0
  Le module xlrd extrait des données d'un tableur Excel à partir de la version 2.0 et les formatter.
- biopython==1.76
  Le package biopython regroupe un ensemble d'outils Python pour le traitement informatique de la biologie moléculaire et comprend le module numpy.
- numpy==1.18.2 (installé avec le package biopython)

- DateTime==4.3
  Le module datetime permet de manipuler des dates et heures en gérant des objets de type DateTime.
- pytz==2019.3 (installé avec le module datetime)
- zope.interface==5.0.1 (installé avec le module datetime)

En outre, certains modules sont déjà présents dans le package standard Python3 :

- le module os fournit une manière portable d'utiliser les fonctionnalités dépendantes du système d'exploitation,
- le module pickle permet de sérialiser et désérialiser une structure d'objet Python. Il remplace le module primitif marshal. pickle se trouve déjà dans la librairie standard Python3,
- le module csv implémente des classes pour lire et écrire des données liées à des feuilles de calcul ou des bases de données au format csv,
- le module shutil propose des opérations sur les fichiers et collections de fichiers, notamment la copie et suppression de fichiers.

Notre package, une fois créé, devrait proposer le même environnement de travail et donc contenir ces mêmes modules.

3.2. Choix de l'outil d'empaquetage

Le Python Packaging User Guide PyPA recommande l'utilisation de :

- *pip* pour l'installation de librairies à partir de Python Package Index PyPI,
- *setuptools* pour définir des projets et créer des sources de distribution,
- *pipenv* pour la gestion des dépendances de librairies lors de développement d'applications Python,
- *venv* pour isoler les dépendances particulières d'une application,
- *conda* pour les projets scientifiques.
- *buildout* pour les projets de développement Web,
- *poetry* pour un besoin particulier non couvert par Pipenv.

pipenv est un gestionnaire de haut niveau pour les environnements, les dépendances et les packages Python. Contrairement à *virtualenv*, *pipenv* distingue les dépendances du projet et les dépendances des dépendances du projet. Par ailleurs, *pipenv* différencie le mode développement du mode production. Il offre l'avantage de bien fonctionner sur Windows. Toutefois, la communauté Python l'a peu mis à jour depuis 2018.

Anaconda est une distribution de logiciels multiplateformes (Windows, Linux, MacOS) qui facilite l'installation des librairies scientifiques *Numpy* et *Scipy*, ce qui est particulièrement intéressant dans le cas des plateformes Windows où ce processus est plus complexe. Elle incorpore une librairie open-source appelée conda permettant la gestion des dépendances, de l'environnement de travail ainsi que la création de packages. *Anaconda* semble être approprié au projet, mais c'est une distribution trop lourde pour être intégrée à notre package et *Miniconda*, qui ne comporte que Python, conda et pip, ne répond pas au besoin du projet.

Nous avons naturellement cherché à construire le package manuellement, à partir de pipenv puis de conda. Cet effort s'est avéré laborieux et a révélé des incompatibilités qui n'ont pas permis de valider les exigences de la plateforme testPyPI.

Nous avons donc décidé de construire notre package en utilisant *poetry*, qui est un outil complet autour duquel la communauté Python reste très active. Il propose à la fois la gestion des dépendances, l'empaquetage (création d'une structure pour un projet et la génération de fichiers de configuration et de manifestes) et la publication.

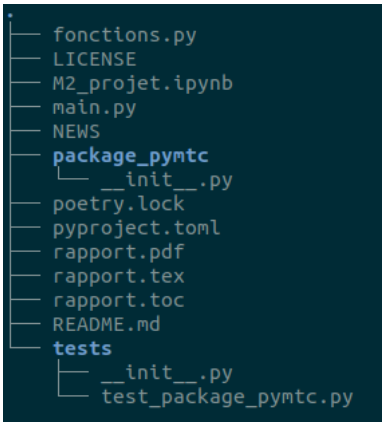
4. La composition du package

4.1. Composition d'un package standard

Un package comporte traditionnellement les fichiers __init__.py, setup.py, LICENSE et README.md.

setup.py est le script de construction destiné à setuptools. Il définit le nom et la version du package, ainsi que les fichiers qu'il contient. Avec la PEP-518, la Python Packaging Authority a proposé un nouveau standard au format package.toml, qui remplace les fichiers setup.py, requirements.txt, setup.cfg, MANIFEST.in et Pipfile. C'est ce standard qui est utilisé lors de la création d'un package avec Poetry.

La commande poetry new nomPaquet permet de générer le squelette de l'application, comprenant les tests unitaires, le fichier pyproject.toml, le fichier README.rst que nous changeons au format markdown README.md, le répertoire du projet et le fichier init.py. Nous rajoutons un fichier LICENSE, les composants principaux de la librairie, le présent rapport de stage et un fichier .gitignore pour la gestion des versions.



La plupart des systèmes d'exploitation incorporent Python2.7 par défaut. L'environnement de travail devra donc expressément définir Python3 comme version pour le projet. Nous avons choisi la version 3.6.5 de Python dans le fichier package_mymtc.toml.

Les paquets construits pour des systèmes Unix (Linux et MacOS) nécessitent l'incorporation de fichiers build.sh et meta.yaml. Les paquets construits pour les systèmes Windows nécessitent l'incorporation des fichiers bld.bat et meta.yaml. A VERIFIER DANS LE CAS DE POETRY

4.2. Composition du package pymtc

4.3. Les différents modules utilisés par le package pyMTC

5. La création de la librairie

=== A CHANGER === Pour créer une librairie à partir de conda, il est nécessaire d'installer conda-build puis de construire un recipe composé de :

- un fichier meta.yaml contenant toutes les métadata du recipe
- un script build.sh qui installe les fichiers de la librairie sur Linux et macOS, exécuté avec une commande bash
- un script bld.bat qui installe les fichiers de la librairie sur Windows, exécuté avec une commande cmd
- un fichier optionnel $run_test.py$, qui s'exécute automatiquement pour effectuer des tests
- un fichier readme et des icônes si nécessaire.

Les trois premiers fichiers se créent avec la commande conda skeleton. ===FIN CHANGEMENT===

6. Quelle licence choisir ?

Trois licences retiennent notre attention. En voici les principales caractéristiques :

- la licence MIT, courte et permissive, préserve exclusivement le copyright et les avis de licence. Toute modification ultérieure peut être distribuée suivant une licence différente et notamment utilisée à des fins personnelles ou commerciales, sans obligation de publication des codes source,
- la licence Apache (2.0) est également permissive et sensiblement similaire dans ses conditions à la licence MIT. Toutefois, elle requiert de préciser les modifications effectuées lors de nouvelles distributions,
- la licence GNU (GPL v3.0) préserve également le copyright et les avis de licence. Elle peut être utilisée à des fins personnelles et commerciales. Elle impose en outre, en cas de modification, la publication complète des codes et l'utilisation de la licence GNU pour les nouvelles distributions.

Nous choisissons la licence MIT qui semble répondre aux besoins de ce projet.

7. Les principaux éléments du code

dico est composé de la façon suivante :

Un fichier pkl tel que dico_africanum.pkl est créé par pickle et contient un flux d'octets représentant les objets à sérialiser.

pickle permet aux objets d'être sérialisés en fichiers sur disque et désérialisés dans le programme au moment de l'exécution.

Références

https://packaging.python.org/guides/
https://realpython.com/pypi-publish-python-package/