uploader un module sur PyPI (Pytbon Package Index) avec distutils (Distributing Python Modules):

Introduction a distutils:

Dans ce tutoriel, issue de mon expérience et dans le but de distribuer une de mes créations, vous apprendrez à uploader un module compatible avec Linux et Windows sur PyPI (Python Package Index) grâce au module distutils dont vous appendrez le stricte minimum grâce a une unique fonction : la fonction setup() afin d'atteindre le but énoncé.

Nous allons dans un premier temps "packager" le module et les fichiers connexes : (README.txt, Licence.txt etc.),grâce au module distutils (Distributing Python Module) de la librairie standard, dont le but est de faciliter la distribution de modules par divers formats de fichiers : -*.zip, *.tar.gz, *.tar.bz2, *.tar.Z et *.tar.

Automatisant l'installation du module dans le répertoire adéquat quelque soit le système, pourvus qu'il dispose d'une implémentation python.

Bien sûr le fichier *.py du module devra être aussi universel mais on peut limiter la distribution à un des systèmes spécifiques et donc le format de sortie adéquat. distutils permet aussi entre autre d'inclure des fichiers de tout types comme des extensions écrit en C ou C++ avec des options de préprocesseur, des fichiers de données comme des images, des scripts et bien sûr des fichiers de documentation (README.txt) et la licence.

Mais distutils permet aussi de générer des fichiers binaires avec la commande bdist (binary distribution) : -*.rpm, *.srpm, *.exe, *.msi.

Puis nous allons uploader le module sur PyPI.

Packaging d'un premier exemple:

Concernant le packaging qui peut être une vocation pour certains (par ex. un membre d'une équipe spécialisée dans ce domaine effectue cette tache avec joie pour son équipe), nous allons commencer avec un exemple simple:

Nous allons réaliser le paquetage d'un module composé d'un seul fichier: foo.py grâce a la fonction setup() de distutils Nous allons ajouter un fichier README.txt automatiquement détecté par distutils. Nous allons ajouter un fichier MANIFEST.in afin d'inclure le fichier License.txt

Fichier foo.py:

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

print "you win"
```

Fichier MANIFEST.in:

```
include License.txt
```

Fichier setup.py:

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

from distutils.core import setup

setup(name='foo',
# Métadonnées: nom du module.
version='1.0',
# Métadonnées: version du module au format major.minor[.patch[.sub]].
author="Luke Spywoker",
# Métadonnées: auteur(s) du mdoule
author_email="Luke-Spywoker@yahoo.com",
# Métadonnées: adresse mail de l'auteur
url="http://www.module_foo.html",
# Métadonnées: URL du module (obligatoire pour upload sur PyPI !)
description="module foo",
# Métadonnées: desription courte <= 200 caractères.
long_description='''module foo for the demonstration
of the module package creation,
and upload on PyPI.'''
# Métadonnées: description longue utiliser par PyPI pour
# la page internet *.html du mdoule sur le
# site de PyPI.
platforms=["Linux","Windows"],
# Métadonnées: liste des platformes cibles.
license="GPL v3",
# Métadonnées: License du module.

py_modules=['foo']) # Le fichier ou la liste des fichiers du module,
                    # sans l'extension '.py'.
```

Puis nous allons vérifier le listing des fichiers manipulés par distutils. Grâce a l'option d'exécution du fichier setup.py --manifest-only qui génère un fichier de listing nommé MANIFEST, le processus peut être inversé en éditant un fichier MANIFEST.in qui liste les fichiers grâce a des directives (à inclure ou à

exclure) avec utilisation de méta-caractères dans l'arborescence que nous avons utilisé pour inclure le fichier Licence.txt.

Notre arborescence contient donc:

```
$ ls
foo.py
License.txt
MANIFEST
MANIFEST.in
README.txt
setup.py

$ python setup.py --manifest-only
$ cat MANIFEST
# file GENERATED by distutils, do NOT edit
License.txt
README.txt
foo.py
setup.py
```

Tout est OK, nous allons générer le paquetage avec un format de fichier *.zip.

```
$ python setup.py sdist --formats=zip
running sdist
running check
reading manifest template 'MANIFEST.in'
writing manifest file 'MANIFEST'
creating foo-1.0
making hard links in foo-1.0...
hard linking License.txt -> foo-1.0
hard linking README.txt -> foo-1.0
hard linking foo.py -> foo-1.0
hard linking setup.py -> foo-1.0
creating dist
creating 'dist/foo-1.0.zip' and adding 'foo-1.0' to it
adding 'foo-1.0/License.txt'
adding 'foo-1.0/README.txt'
adding 'foo-1.0/setup.py'
adding 'foo-1.0/PKG-INFO'
adding 'foo-1.0/foo.py'
removing 'foo-1.0' (and everything under it)
```

Le traceback de la commande nous renseigne la création du fichier zip dans le sous-dossier dist.

Pour vérifier le bon fonctionnement de distutils, c'est-à-dire l'installation de notre module foo.py, et comme le ferai l'utilisateur nous décompressons l'archive et installons le module en exécutant le fichier setup.py avec comme argument install en tant qu'utilisateur root.

NB: Cette procédure est automatisée si le module provient de PyPI.

Installation:

Notre paquetage est prêt a être distribué et l'installation sur le système de destinations se fait en exécutant le fichier setup.py avec comme argument install.

```
$ cd dist
$ ls
foo-1.0.zip
$ unzip foo-1.0.zip
Archive:  foo-1.0.zip
foo-1.0/License.txt
foo-1.0/README.txt
foo-1.0/setup.py
foo-1.0/PKG-INFO
foo-1.0/foo.py
$ ls
foo-1.0  foo-1.0.zip
$ cd foo-1.0
$ ls
foo.py  License.txt  PKG-INFO  README.txt  setup.py
$ sudo python setup.py install
running install
running build
running build_py
creating build
creating build/lib.linux-x86_64-2.7
copying foo.py -> build/lib.linux-x86_64-2.7
running install_lib
copying build/lib.linux-x86_64-2.7/foo.py -> /usr/local/lib/python2.7/dist-packages
byte-compiling /usr/local/lib/python2.7/dist-packages/foo.py to foo.pyc
running install_egg_info
Writing /usr/local/lib/python2.7/dist-packages/foo-1.0.egg-info
```

Le traceback nous renseigne que le module a été installé dans le dossier /usr/local/lib/python2.7/dist-packages/ et la création d'un fichier foo-1.0.egg-info qui contient les métadonnées du module.

Vérifions maintenant que tout est OK concernant l'installation du module.

```
$ cat /usr/local/lib/python2.7/dist-packages/foo-1.0.egg-info
Metadata-Version: 1.0
Name: foo
Version: 1.0
Summary: module foo
Home-page: http://www.module_foo.html
Author: Luke Spywoker
Author-email: Luke-Spywoker@yahoo.com
License: GPL v3
Description: module foo for the demonstration
    of the module package creation,
    and upload on PyPI.
Platform: Linux
Platform: Windows
$ python
Python 2.7.3 (default, Sep 26 2013, 20:03:06)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import foo
```

```
you win
>>> quit()
```

Il ne reste plus qu'à uploader notre module sur PyPI. Mais il y a un problème qui concerne notre portabilité sous système Windows car si cette méthode fonctionne parfaitement pour Linux, sous Windows il aurai fallut taper:

```
C:\\Python27> python C:/Users/../foo-1.0/foo-1.0/setup.py install
running install
running build
running build_py
file foo.py (for module foo) not found
file foo.py (for module foo) not found
running install_lib
warning: install_lib: 'build\\lib' does not exist -- no Python modules to install

running install_egg_info
Writing C:\\Python27\\Lib\\site-packages\\foo-1.0-py2.7.egg-info
```

Et le seule fichier crée est le fichier foo-1.0-py2.7.egg-info dans le dossier C:\Python27\Lib\site-packages\ en théorie si distutils avait copié le fichier foo.py dans C:\Python27\Lib\site-packages\ cela aurai parfaitement fonctionné. Je ne sais pourquoi mais distutils a ses raisons que la raison ne connaît pas.

Packaging d'un module compatible Linux et Windows:

Introduction:

Par contre, je vais vous montrer (avec un exemple plus conséquent et l'upload final sur PyPI) comment générer un module avec des fichiers d'exemple d'utilisation du module et des techniques concernant la page *.html de notre module sur le site de PyPI grâce au format de fichier *.rst.

A savoir que notre tentative a échoué car d'après moi Windows exige la création d'un dossier dans C:\Python27\Lib\site-packages\ pour notre module avec un fichier __init__.py dedans, ce que je pense aussi plus adapté a la distribution d'un module avec donc: dans le dossier les fichiers *.py et __init__.py et des sous-dossiers /doc pouvant contenir simplement le README.txt pour la documentation et un autre sous dossier /License.

Le code du fichier __init__.py peut être vierge servant à détecter, dans l'arborescence notre fichier du module, le fichier __init__.py est exécuté quand ont importe le module et peut contenir entre autres du code renseignant les parties d'un package de fichiers *.py se reportant au classes fonctions d'un module plus vaste dans l'arborescence du module.

Mais dans mon cas, je copies-collé le contenus de mon fichier de mon module dans le fichier __init__.py ce qui est une pratique nécessaire au bon fonctionnement de l'import.

Un exemple complet:

Pour commencer, voici l'arborescence du dossier de mon package, pour la distribution de mon module "curser" dans le dossier curser_package, qui contient donc aussi un setup.py, un README.txt, License.tx, etc...

```
curser_package/
curser_package/MANIFEST.in
curser_package/README.rst
curser_package/README.txt
curser_package/setup.py
curser_package/src/
curser_package/src/curser/
curser_package/src/curser/Doc/
curser_package/src/curser/Doc/README.txt
curser_package/src/curser/Examples/
curser_package/src/curser/Examples/curser_example_face.py
curser_package/src/curser/Examples/curser_example_fractals.py
curser_package/src/curser/Examples/curser_example_koch.py
curser_package/src/curser/Examples/curser_example_spirals.py
curser_package/src/curser/License/
curser_package/src/curser/License/License.txt
curser_package/src/curser/__init__.py
curser_package/src/curser/curser.py
```

Commençont par le fichier setup.py:

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

from distutils.core import setup

with open("README.rst",'r') as file :
  long_description = file.read()

setup(name='curser',version='1.0.0',
url='https://github.com/mrcyberfighter/curser',
author="Eddie Bruggemann",author_email="mrcyberfighter@gmail.com",
maintainer="Eddie Bruggemann",maintainer_email="mrcyberfighter@gmail.com",

long_description=long_description,
description='''curser is a module based and complementary to pygame:
an turtle implementation for the pygame module.
With appeareance,control,drawing,orientation and coordinates retrieving functions.''',

packages=['curser'],
package_dir={'curser': 'src/curser'},
package_data={'curser': ['curser/*.py','Doc/*.txt','License/*.txt']},

data_files=
[('curser',["src/curser/Doc/README.txt","src/curser/License/License.txt"]),
 ('curser/Examples',["src/curser/Examples/curser_example_face.py",
                     "src/curser/Examples/curser_example_fractals.py",
                     "src/curser/Examples/curser_example_koch.py",
                     "src/curser/Examples/curser_example_spirals.py"])
],
platforms=["Linux","Windows"],license="GPL3")
```

Explications:

Je vais vous expliquer au fur et a mesure la signification des arguments que nous n'avons pas encore vue et le sens de l'argument "long_description" qui a pour valeur le contenu d'un fichier *.rst qui est l'extension correspondante au langage ReStructuredText et qui est composé de schémas logiques. Ces schémas sont interprétés par un convertisseur HTML pour produire "Very Structured Text" (un texte très structuré) qui pourra être utilisé par un navigateur web. Le tout est utilisé par PyPI pour générer la page *.html de votre module et est également utilisé par github pour afficher la description d'un dépôt entre autres formats de ces langages de schémas logiques qui sont visualisable de manière claire en mode texte aussi bien que convertie en *.html.

Pour créer le fichier README.rst j'ai utilisé l'éditeur de texte retext qui permet un aperçus du résultat en *.html et le programme rst2html du paquet python-docutils et qui effectue la conversion d'un fichier *.rst en fichier *.html.

-) L'argument packages de la fonction setup de distutils.core permet d'indiquer le nom du package, comme une liste de string.

-) L'argument package_dir permet de définir la racine du package donc du dossier qui sera crée dans le dossier de destination ou distutils va copier les fichiers comme clefs du dictionnaire et comme valeur le dossier local correspondant.

-) L'argument package_data permet de lister les fichiers que l'on désire copier, avec comme clefs du dictionnaire le nom du package et comme valeur la liste des fichiers a inclure.

Attention : il faut les lister car distutils ne fait pas de recherche récursive mais supporte les méta-caractères comme '*'.

-) L'argument data_files permet de lister les fichiers de données, sous forme de liste de tuple (dossier, fichier), de tous types (images, données, etc.) qui seront copiés automatiquement par distutils dans un autre dossier que les fichiers code source du module, adapté a cet effet.

Voila, maintenant vous savez tout ce que j'ai pu comprendre des entrailles de distutils.core.setup().

Il vous suffit de taper:

```
$ python setup.py sdist
```

avec éventuellement un argument:

--manifest-only afin de vérifier la bonne prise en charge des fichiers par distutils.

--formats afin de spécifier le format de sortie.

Sinon la procédure est la même que celle que nous avons vus précédemment.

Upload sur PyPI:

Inscription:

Pour s'inscrire sur PyPI (ce qui est nécessaire si vous désirez uploader un module distutils), nous allons simplifier le travail car il suffit d'exécuter le fichier setup.py avec comme argument register.

```
$ python setup.py register
running register
We need to know who you are, so please choose either:
1. use your existing login,
2. register as a new user,
3. have the server generate a new password for you (and email it to you), or
4. quit
Your selection [default 1]:
```

Si vous n'avez pas encore un compte choisissez 1, suite a quoi un nom d'utilisateur ainsi qu'un mot de passe associé vous sera demandé ainsi qu'une adresse mail a laquelle vous sera envoyé un lien pour finaliser votre inscription.

NB: Pendant ce processus il vous sera demandé d’accepter les conditions de renonciation a toute revendication de propriété sur les modules uploadés ainsi que de donner le droit à PyPI de modifier les fichiers - ce qui est bien sûr gênant, mais je pense que cela ne concerne uniquement la correction d'éventuelles erreurs dans le README plutôt que la modification du code source de votre module.

Upload du module:

Pour cela il suffit d'exécuter votre fichier setup.py avec l'argument upload.

```
$ python setup.py sdist --format=zip upload
...
# Demande de votre nom d'utilisateur et mot de passe.
# Il vous sera également demander comme c'est la deuxième fois
# que nous nous connectons a PyPI la création d'un fichier .pypirc
# sur votre machine,
# qui contiendra votre couple nom d'utilisateur mot de passe
# en clair (a vous de voir) et
# le serveur sur lequel votre module sera héberger, car vous pouvez
# l'héberger sur le serveur que vous voulez le serveur par défaut étant
# celui de PyPI noter pypi dans le fichier, avec l'argument
# supplémentaire --repository=url.
```

Voila, c'est tout.

Sources et Remerciements:

Sources:

distutils:	How-To.
Docutils:	Documentation Utilities.
RestructeredText:	Documentation.
RestructeredText:	Quickstart (fr).

Remerciement:

Auteur:	Bruggemann Eddie
Corrections:	Pinta Oleander
Date:	2013