Setuptools

En Python todo el tema de empaquetar puede ser un poco lioso, ya que encontramos varios módulos desintados a ello. Nosotros vamos a centrarnos en **setuptools**, ya que es la forma más utilizada, nos proporciona todo lo necesario para distribuir nuestros propios módulos e incluso nos permite publicar paquetes en el respositorio público PyPI (Python Package Index) de forma directa desde la propia terminal.

Si lo recordáis, en la lección de módulos ya os enseñé como crear un distribuible con setuptools, a lo largo de esta lección vamos a repasar y aprender varios conceptos nuevos.

Paquete básico

Antes de comenzar es importante repasar la estructura de un paquete en Python,ya que para distribuir nuestro código es indispensable estructurarlo dentro de un paquete:

```
| setup.py         # Fichero que contiene toda la información de instalación
+ prueba/          # Directorio del paquete al mismo nivel que setup.py
   | __init__.py   # Fichero que indica que el directorio es un paquete
   | modulo.py     # Módulo o script que contiene definiciones
```

Por lo tanto vamos a empaquetar el paquete de nombre **prueba**, que contiene código en el fichero *modulo.py*.

Vamos a aprender un poco más sobre el fichero de instalación.

El fichero de configuración incluye toda la información necesaria para realizar la instalación de nuestro paquete. Algunos campos incluyen sólo metadatos como el nombre, la versión, la descripción o el autor. Pero otros sirven para extender la instalación.

Como sería un caos que cada desarrollador pusiera los campos que quisiera, hay una serie de parámetros comunes y avanzados, pero como son muchos lo más común es utilizar una plantilla base como la siguiente que pasa la configuración a la función **setup**:

setup.py

```
from setuptools import setup

setup(name="Prueba",  # Nombre
    version="0.1",  # Versión de desarrollo
    description="Paquete de prueba",  # Descripción del funcionamiento
    author="Hector Costa",  # Nombre del autor
    author_email='me@hcosta.info',  # Email del autor
    license="GPL",  # Licencia: MIT, GPL, GPL 2.0...
    url="http://ejemplo.com",  # Página oficial (si la hay)
    packages=['prueba'],
)
```

¿Hasta aquí fácil no? Son simples metadatos para definir el paquete, con la excepción de **packages**, en el que tenemos que indicar todos los paquetes que formarán parte del paquete distribuido en forma de lista.

Aunque en este caso únicamente tendríamos al paquete **prueba**, imaginaros que tenemos docenas de subpaquetes y tubiéramos que añadirlos uno a uno... Pues para estos casos podemos importar una función que se encargará de buscar automáticamente los subpaquetes, se trata de **find_packages** y la podemos encontrar dentro de **setuptools**:

```
from setuptools import setup, find_packages

setup(...
      packages=find_packages()
)
```

Ahora imaginaros que en vuestro paquete algún código utiliza funciones de un módulo externo o paquete que hay que instalar manualmente. Esto se conoce como dependencias del paquete, y por suerte podemos indicar a un parámetro que descargue todos los paquetes en la versión que nosotros indiquemos, se trata de **install_requires**.

Por ejemplo imaginad que dentro de nuestro paquete necesitamos utilizar el módulo **Pillow** para manejar imágenes. Por regla general podemos instalarlo desde la terminal con el comando:

```
pip install pillow
```

Pero si queremos que el paquete lo instale automáticamente sólo tenemos que indicarlo de esta forma:

```
setup(...,
      install_requires=["pillow"],
)
```

Y así iríamos poniendo todas las dependencias en la lista.

Lo bueno que tiene es que podemos indicar la versión exacta que queremos instalar, por ejemplo. Si mi programa utilizase la versión 1.1.0 de Pillow tendría que poner:

```
setup(...,
      install_requires=["pillow==1.1.0"],
)
```

En cambio si fuera compatible con cualquier versión a partir de la 1.1.5 podría poner:

```
setup(...,
      install_requires=["pillow>=1.1.5"],
)
```

Si no indicamos una versión, se instalará automáticamente la más actual.

De forma similar a antes, quizá llega el momento donde tenemos muchísimas dependencias y es un engorro tener que cambiar directamente el fichero **setup.py**. Para solucionarlo podemos utilizar una técnica que se basa en crear un fichero de texto y escribir las dependencias, una por línea.

Luego podemos abrir el fichero y añadir las dependencias automáticamente en forma de lista. Generalmente a este fichero se le llama **requirements.txt** y debe estar en el mismo directorio que **setup.py**:

requirements.txt

```
pillow==1.1.0
django>=1.10.0,<=1.10.3
pygame
```

Luego en las dependencias indicaríamos lo siguiente:

```
setup(...,
      install_requires=[i.strip() for i in open("requirements.txt").readlines()],
)
```

Suite Test

Otra cosa interesante que podemos hacer es adjuntar una suite de tests unitarios para nuestro paquete, ya sabéis, los que aprendimos en la unidad anterior.

Para incluirlos tendremos indicar un parámetro en el instalador llamado **test_suite**, al que le pasaremos el nombre del directorio que los contiene, por lo general llamado **tests**:

```
   | __init__.py
   | modulo.py
+ tests/
   | __init__.py
   | test_pillow.py
   | test_django.py
   | test_pygame.py
```

En el **setup.py**:

```
setup(...,
      test_suite="tests"
)
```

Luego para ejecutarlos podemos utilizar el comando:

```
python setup.py test
```

PyPI y PIP

Por último hablemos un poco más del **Python Package Index**.

Como ya sabéis se trata de un repositorio público con miles y miles de paquetes creados por la enorme comunidad de Python.

La forma de instalar cómodamente los paquetes de PyPI es con la herramienta PIP (un acrónimo recursivo de Pip Installs Packages), utilizando el comando **pip install nombre_paquete**.

Además podemos listar los paquetes instalados con **pip list**, borrar alguno con **pip uninstall nombre_paquete** o incluso instalar todas las dependencias de un fichero **requisites.txt** utilizando **pip install requisites.txt**.

Si queréis saber más sobre pip, simplemente escribid **pip** en la terminal.

Clasificadores

Por lo tanto tenemos un repositorio inmenso, así que ¿cómo podemos añadir información para categorizar nuestro paquete en PyPI? Pues utilizando un parámetro llamado **classifiers** de la siguiente forma:

```
setup(...,
    classifiers=[
        "Development Status :: 3 - Alpha",
        "Topic :: Utilities",
        "License :: OSI Approved :: GNU General Public License (GPL)",
    ],
)
```

Hay un montón de clasificadores, desde el estado del proyecto, el tema, las licencias, etc. Una lista completa de los clasificadores disponibles podemos encontrarla en la propia web de PyPI: https://pypi.python.org/pypi?%3Aaction=list_classifiers

Probando el paquete

Una vez tenemos toda la información configurada, podemos probar nuestro paquete fácilmente realizando una instalación en modo desarrollo. Para ello utilizaríamos el siguiente comando:

```
python setup.py develop
```

Este modo es muy práctico, ya que nos permite utilizar nuestro módulo en cualquier lugar y hacer modificacione sin necesidad de reinstalarlo constamente. Eso es posible porque se utiliza desde el propio directorio.

Una vez hayamos hecho las probaturas y estemos satisfechos, podemos desinstalar el paquete de desarrollo:

```
python setup.py install
```

Pero tenemos que tener en cuenta que una vez hecho esto, el paquete se instala en una copia interna y ya no podremos modificarlo sin antes desinstalarlo, algo que tendremos que hacer con PIP, buscando el nombre del paquete con **pip list** y haciendo un **pip uninstall nombre_paquete**.

Distribuyendo el paquete

Ya tenemos el paquete, hemos creado el instalador, lo hemos probado y estamos preparados para distribuirlo. Hay dos formas:

- **Localmente**: Generando un fichero comprimido que podemos compartir con nuestros conocidos.
- **Públicamente**: En el repositorio PyPI para que todo el mundo pueda utilizarlo.

Evidentemente si distribuimos localmente no tenemos que tener mucho cuidado, y además podemos hacer pruebas. Pero si decidimos hacerlo públicamente tendremos que intentar que el paquete tenga un mínimo de calidad.

Localmente

Distribuir el paquete localmente es muy fáci. Simplemente tenemos que utilizar el comando:

```
python setup.py sdist
```

Esto generará un directorio **dist/** en la carpeta del paquete. Dentro encontraremos un fichero zip o tar.gz dependiendo de nuestro sistema operativo.

Este fichero ya podremos compartirlo con quien queramos, y para instalarlo sólo tendremos que utilizar la herramienta **pip**:

```
pip install nombre_del_fichero.zip
```

Luego para desinstalarlo de la misma forma pero utilizando el nombre del paquete:

```
pip uninstall nombre_paquete
```

Públicamente

Aunque no voy a hacer la demostración porque ahora mismo no dispongo de un paquete para publicar en el repositorio de PyPI, sí que os voy a enseñar los pasos a seguir para hacerlo. Lo bueno de registrar un paquete en PyPI es que podemos instalarlo desde cualquier lugar a través de internet utilizando la herramienta PIP.

Dicho ésto, si algún día creáis un paquete de calidad y queréis compartirlo con la comunidad, lo primero es registrar una cuenta en PyPI [https://pypi.org/account/register/].

A continuación desde el directorio de nuestro paquete tenemos que ejecutar el comando:

```
python setup.py register
```

Así iniciaremos una petición para registrar nuestro paquete en el repositorio. Luego tendremos que seguir los pasos e identificarnos cuando lo pida con nuestro usuario y contraseña (que hemos creado antes).

Una vez hecho esto ya hemos creado nuestro paquete, pero todavía no hemos publicado una versión, así que vamos a hacerlo utilizando el comando:

```
python setup.py sdist upload
```

¡Y ya está! Ahora podremos instalar nuestro paquete desde en cualquier lugar con PIP:

```
pip install nombre_paquete
```