

# **Diplomatura en Python:**

# **Python nivel intermedio**

Módulo 1:

# Nivel Intermedio I

Unidad 2:

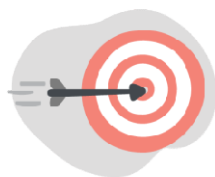
## Módulos II



## Presentación

En esta unidad profundizaremos en la creación de módulos y paquetes y veremos cómo implementar la herramienta opcional setuptools, la cual nos permitirá contar con más opciones a la hora de la creación de nuestra distribución.

También veremos cómo trabajar con entornos virtuales, lo cual puede ser una opción muy interesante si no nos interesa modificar nuestra distribución de versión por defecto.



## Objetivos

Aprendan a crear un paquete básico.

Profundicen en la creación del archivo setup.py utilizado en la instalación de paquetes.

Aprendan a trabajar con entornos virtuales.



## Bloques temáticos

- 1.- Crear una distribución.
- 2.- Archivo setup.py
- 3.- Creamos un paquete mediante setuptools.

# 1. Crear una distribución

Crearemos a continuación una distribución simple para comprender los pasos necesarios que debemos seguir, lo haremos siguiendo una serie de pasos simples. Podemos crear módulos para:

A.- Agregarlos a nuestra distribución de Python

B.- Compartir con otros a través de PyPI

## Paso 1 – Creamos el o los archivos a incluir en el paquete (directorio)

Supongamos que creó el archivo prueba1.py y lo guardó dentro del paquete prueba1.

**prueba1**

prueba1.py

prueba1.py

```
def recorrerLista(item, nivel=0):    # Agregar valor por defecto

    for x in item:

        if isinstance(x, list):

            recorrerLista(x, nivel + 1)

        else:

            for y in range(nivel):

                print("\t", end="")    # Agregar indentación en lugar de saltos de línea

            print(x)
```

## Paso 2 – Le agrego un archivo denominado **setup.py**

**prueba1**

prueba1.py

setup.py

El contenido de **setup.py** es el siguiente:

```
from distutils.core import setup

setup(
    name = 'prueba1',
    version = '1.0.0',
    py_modules = ['prueba1'],
    author = 'hfpypython',
    author_email = 'juanbarretor@gmail.com',
    url = 'fascinaweb.com',
    description = 'Es una prueba de uso',
)
```

Estos son los nombres de los argumentos de la función setup

**Nota:** Con py\_modules asociamos los metadatos del módulo con los argumentos de la función setup



## Paso 3 – Construimos la distribución

Entramos al directorio del paquete desde el cmd (en mi caso lo tengo en el escritorio) y escribimos:

```
C:/Users/Marcelo/Desktop/prueba1>python setup.py sdist
```

Salen:

```
C:\Users\juanb\Desktop\prueba1>python setup.py sdist
running sdist
running check
warning: sdist: manifest template 'MANIFEST.in' does not exist (using default file list)

warning: sdist: standard file not found: should have one of README, README.txt,
README.rst

writing manifest file 'MANIFEST'
creating prueba1-1.0.0
making hard links in prueba1-1.0.0...
hard linking prueba1.py -> prueba1-1.0.0
hard linking setup.py -> prueba1-1.0.0
creating dist
creating 'dist\prueba1-1.0.0.zip' and adding 'prueba1-1.0.0' to it
adding 'prueba1-1.0.0\PKG-INFO'
adding 'prueba1-1.0.0\prueba1.py'
adding 'prueba1-1.0.0\setup.py'
```



removing 'prueba1-1.0.0' (and everything under it)

Ahora la estructura quedó de la siguiente forma:

```
prueba1
  dist
    prueba1-1.0.0.zip
      prueba1-1.0.0
        PKG-INFO
        prueba1.py
        setup.py
      MANIFEST
      prueba1.py
      setup.py
```

dist dentro tiene un archivo .zip con nombre igual al paquete y al cual se le ha agregado la versión.



## Paso 4 – Instalamos la distribución dentro de nuestra distribución local de Python.

Escribimos en el cmd:

```
C:\Users\Marcelo\Desktop\prueba1>python setup.py install
```

Nos retorna:

```
C:\Users\juanb\Desktop\prueba1>python setup.py install
running install
running build
running build_py
creating build
creating build\lib
copying prueba1.py -> build\lib
running install_lib
copying build\lib\prueba1.py -> C:\Python37-32\Lib\site-packages
byte-compiling C:\Python37-32\Lib\site-packages\prueba1.py to prueba1.cpython-37.pyc
running install_egg_info
Writing C:\Python37-32\Lib\site-packages\prueba1-1.0.0-py3.7.egg-info
```

Ahora la estructura quedó de la siguiente forma:

```
prueba1
  build
    lib
      prueba1.py
    dist
      prueba1-1.0.0.zip
        prueba1-1.0.0
          PKG-INFO
          prueba1.py
          setup.py
MANIFEST
prueba1.py
setup.py
```

## Paso 5 – Ejecutamos módulo.

Nuestro módulo ahora se puede ejecutar desde el paquete creado que se encuentra en el directorio python1 ó desde la copia que se creó en la distribución de python que tenemos en nuestra máquina en el archivo:

**C:\Python37\Lib\site-packages**

Aquí se crearon los archivos: “prueba1.py” y “prueba1-1.0.0-py3.7.egg-info”

**prueba1-1.0.0-py3.7.egg-info** tiene el mismo contenido que PKG-INFO

además dentro de:

**C:\Python34\Lib\site-packages\\_\_pycache\_\_**

Se creó el binario de nuestro módulo.

Si el módulo lo ejecutamos desde la distribución de python en nuestra máquina el código sería:

```
main.py
```

```
import prueba1
```

```
lista = ["elemento1n1", "elemento2n1", "elemento3n1",
```

```
["elemento1n2", "elemento2n2", "elemento3n2",
```

```
["elemento1n3", "elemento2n3", "elemento3n3"]]]
```

```
prueba1.recorrerLista(lista)
```

Retorna:

```
elemento1n1
elemento2n1
elemento3n1
    elemento1n2
    elemento2n2
    elemento3n2
        elemento1n3
        elemento2n3
        elemento3n3
```

## Paso 6 –Modificaciones posteriores.

Si realizáramos una modificación en el código, para que esta se grave tendríamos que ejecutar el comando:

```
C:\Users\Marcelo\Desktop\prueba1>python setup.py install
```

## 2. Archivo Setup.py

El script de configuración es el centro de toda actividad en la construcción, distribución e instalación de módulos utilizando los **Distutils (Mecanismos de distribución de módulos y paquetes)**. El propósito principal de la secuencia de comandos de configuración es describir la distribución de su módulo a los Distutils, para que los diversos comandos que operan en sus módulos hagan lo correcto. El script de configuración consiste principalmente en una llamada a `setup()`, y la mayoría de la información suministrada a los Distutils por el desarrollador del módulo se proporciona como argumentos clave para `setup()`.

### Creamos un paquete.

Consideremos un ejemplo un poco más complejo en el cual incluimos un par de paquetes según la siguiente estructura:

```
Ejemplo paquete
ejemplo_paquete
command
__init__.py
main1.py
main2.py
__init__.py
main.py
setup.py
```

Es importante que dentro de cada directorio (que consideraremos como paquete) exista un archivo `__init__.py` aun cuando esté vacío.

Nuestro archivo setup.py queda como:

setup.py

```
from distutils.core import setup

setup(
    name = 'prueba1',
    version = '1.0.0',
    py_modules = ['prueba1'],
    author = 'hfpython',
    author_email = 'juanbarretor@gmail.com',
    url = 'fascinaweb.com',
    description = 'Es una prueba de uso',
    packages=['ejemplo_paquete', 'ejemplo_paquete.command'],
)
```

Al instalar el paquete podemos ir al paquete instalado dentro de **sites\_packages/ejemplo\_paquete** y ver que dentro de cada directorio se ha creado un directorio llamado `__pycache__` dentro del cual se han adjuntado los archivos que contienen con extensión “.pyc” con el nombre seguido de la versión de python utilizada.

De esta forma podemos ingresar a un archivo y utilizar las funciones que se encuentran dentro de cada módulo importado.

ejemplo.py

```
from ejemplo_paquete import main
```



```
from ejemplo_paquete.command import main1, main2
```

```
lista = ["elemento1n1", "elemento2n1", "elemento3n1",  
["elemento1n2", "elemento2n2", "elemento3n2",  
["elemento1n3", "elemento2n3", "elemento3n3"]]]
```

```
main.recorrerLista(lista)
```

```
main1.recorrerLista(lista)
```

```
main2.recorrerLista(lista)
```



## 3. Creamos un paquete mediante Setuptools.

Setuptools es una colección de mejoras a los “distutils” de Python que permiten a los desarrolladores crear y distribuir paquetes de Python más fácilmente, especialmente aquellos que tienen dependencias de otros paquetes. Los paquetes construidos y distribuidos usando setuptools se parecen a los creados utilizando directamente los “distutils”.

Crear paquetes utilizando setuptools presenta muchas mejoras, las cuales pueden ser encontradas en: <https://setuptools.readthedocs.io/en/latest/setuptools.html>

## Instalamos setuptools

Para instalar la última versión de setuptools ejecutamos desde el cmd:

```
pip install -U setuptools --user
```

## Utilización

En lugar de importar setup desde “distutils.core” lo realizamos desde setuptools de la siguiente manera:

```
setup.py

from setuptools import setup, find_packages

setup(
    name="HolaMundo",
    version="0.1",
```

```
packages=find_packages(),  
)
```

Veamos los parámetros utilizados aquí con un poco más de detalle:

## Versión

Setuptools puede funcionar bien con la mayoría de los esquemas de versiones. Sin embargo, hay algunas cosas especiales que debe tener en cuenta para garantizar que setuptools y EasyInstall siempre puedan saber qué versión de su paquete es más nueva que otra versión. Saber estas cosas también nos ayudará a especificar correctamente de qué versiones de otros proyectos depende su proyecto.

Una versión consiste en una serie alterna de números de publicación y etiquetas de publicación previa o posterior. Un número de publicación es una serie de dígitos separados por puntos, como 2.4 o 0.5. Cada serie de dígitos se trata numéricamente, por lo que las versiones 2.1 y 2.1.0 son formas diferentes de deletrear el mismo número de versión. Un 2.1 indicaría la primera modificación de versión dentro de la versión 2 mientras que 2.10 indicaría la décima versión de la versión 2. Los ceros iniciales dentro de una serie de dígitos se ignoran, por lo que 2.01 es lo mismo que 2.1 y diferente de 2.0.1.

Después de un número de versión, puede tener una etiqueta previa o posterior a la liberación. Las etiquetas de prelanzamiento hacen que una versión se considere más antigua que la versión a la que se anexa. Por lo tanto, la revisión 2.4 es más reciente que la revisión 2.4c1, que a su vez es más reciente que 2.4b1 o 2.4a1. Las etiquetas posteriores a la publicación hacen que una versión se considere más nueva que la versión a la que se anexa. Por lo tanto, las revisiones como 2.4-1 y

2.4pl3 son más recientes que 2.4, pero son más antiguas que 2.4.1 (que tiene un número de versión más alto).

Una etiqueta de prelanzamiento es una serie de letras que están alfabéticamente antes de "final". Algunos ejemplos de etiquetas preliminares incluirían alfa, beta, a, c, dev, etc. No tiene que colocar un punto o guión antes de la etiqueta de presentación preliminar si está inmediatamente después de un número, pero puede hacerlo si lo prefiere. Por lo tanto, 2.4c1 y 2.4.c1 y 2.4-c1 representan la versión candidata 1 de la versión 2.4, y se tratan como idénticos por setuptools.

Además, hay tres etiquetas especiales de presentación preliminar que se tratan como si fueran la letra c: pre, preview y rc. Por lo tanto, la versión 2.4rc1, 2.4pre1 y 2.4preview1 son todas exactamente la misma versión que 2.4c1, y son tratadas como idénticas por setuptools.

Una etiqueta posterior a la publicación es una serie de letras que son alfabéticamente mayores o iguales a "final", o un guión (-). Las etiquetas posteriores al lanzamiento generalmente se usan para separar números de parches, números de puerto, números de compilación, números de revisión o sellos de fecha del número de lanzamiento. Por ejemplo, la versión 2.4-r1263 podría denotar la revisión 1263 de Subversion de un parche posterior a la versión 2.4. O puede usar 2.4-20051127 para denotar una publicación posterior con una marca de fecha.

Hay que tener en cuenta que después de cada etiqueta anterior o posterior al lanzamiento, puede colocar otro número de publicación, seguido de nuevo por más etiquetas anteriores o posteriores al lanzamiento. Por ejemplo, 0.6a9.dev-r41475 podría denotar la versión 41475 de Subversion de la versión en desarrollo a9 de la versión 0.6. En este caso dev es una etiqueta de pre lanzamiento, por lo que esta versión es un número de versión inferior a 0.6a9. Pero el -r41475 es una etiqueta posterior al lanzamiento, por lo que esta versión es más reciente que 0.6a9.dev.

En su mayor parte, la interpretación de setuptools de los números de versión es intuitiva, pero podemos seguir las siguientes recomendaciones:

- No debemos agregar etiquetas de pre lanzamiento juntas sin un punto o un número entre ellas. La versión 1.9adev es la versión preliminar de 1.9, no una versión preliminar de desarrollo de 1.9a. En su lugar, debemos utilizar .dev, como en 1.9a.dev, o separar las etiquetas preliminares con un número, como en 1.9a0dev. 1.9a.dev, 1.9a0dev e incluso 1.9.a.dev, estas son versiones idénticas desde el punto de vista de setuptools, por lo que podemos utilizar el esquema con el que nos sintamos más cómodos.

- Si queremos estar seguros de que el esquema de numeración elegido funciona de la manera que pretendemos, podemos utilizar la función **pkg\_resources.parse\_version ()** para comparar diferentes números de versión:

La siguiente es una comparación entre opciones de versiones:

```
>>> from pkg_resources import parse_version  
  
>>> parse_version('1.9.a.dev') == parse_version('1.9a0dev')  
  
True  
  
>>> parse_version('2.1-rc2') < parse_version('2.1')  
  
True  
  
>>> parse_version('0.6a9dev-r41475') < parse_version('0.6a9')  
  
True
```

## find\_packages()

Para proyectos simples, generalmente es bastante fácil agregar paquetes manualmente al argumento de paquetes de configuración (). Sin embargo, para proyectos muy grandes (Twisted, PEAK, Zope, Chandler, etc.), puede ser una gran carga mantener la lista de paquetes actualizada. Para eso es que se utiliza `setuptools.find_packages ()`

`find_packages ()` considera un directorio raíz, y dos listas, una para incluir paquetes y otra para excluirlas. Los paquetes solo se reconocen si incluyen un archivo `__init__.py`.

Los patrones de inclusión y exclusión son nombres de paquetes que, opcionalmente, incluyen comodines. Por ejemplo, `find_packages (exclude = ["*. tests"])` excluirá todos los paquetes cuya

última parte sea “test”. O, `find_packages(exclude = ["*. Tests", "*.tests.*"])` También excluirá cualquier sub paquete de paquetes con el nombre test, pero no excluirá un paquete de pruebas (tests) de nivel superior ni sus hijos. De hecho, si realmente no quieres paquetes de pruebas(tests), necesitarás algo como esto:

```
find_packages(exclude=["*.tests", "*.tests.*", "tests.*", "tests"])
```

Setuptools es una herramienta excelente y utilizada por frameworks como Django, podemos encontrar una referencia muy completa en:

<https://setuptools.readthedocs.io/en/latest/setuptools.html>

Y ver un ejemplo de su utilización si nos descargamos Django en su versión comprimida y analizamos el archivo `setup.py` que trae. Este punto se deja como tarea opcional.



## **Bibliografía utilizada y sugerida**

### **Libros y otros manuscritos:**

**Programming Python 5th Edition – Mark Lutz – O'Reilly 2013**

**Programming Python 4th Edition – Mark Lutz – O'Reilly 2011**

### **Manual online**

**<https://docs.python.org/3.7/tutorial/>**

**<https://docs.python.org/3.7/library/index.html>**

**<https://docs.python.org/3/distutils/introduction.html>**