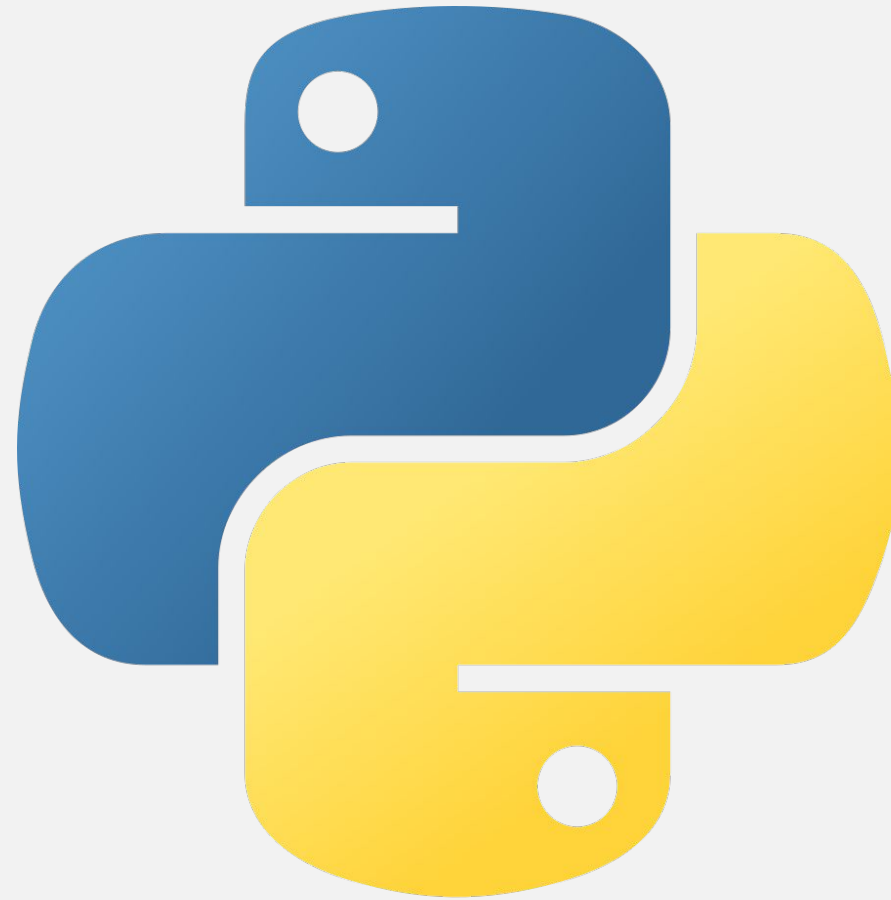


PIP



¿Qué es PIP?

Es un sistema utilizado para instalar y administrar paquetes de software escritos en Python.

PIP es el acrónimo de Paquete de Instalación de Python (Python Package Installer)

A partir de Python 3.4 y Python 2.7.9 viene instalado por defecto.

`pip --version`



Breve historia

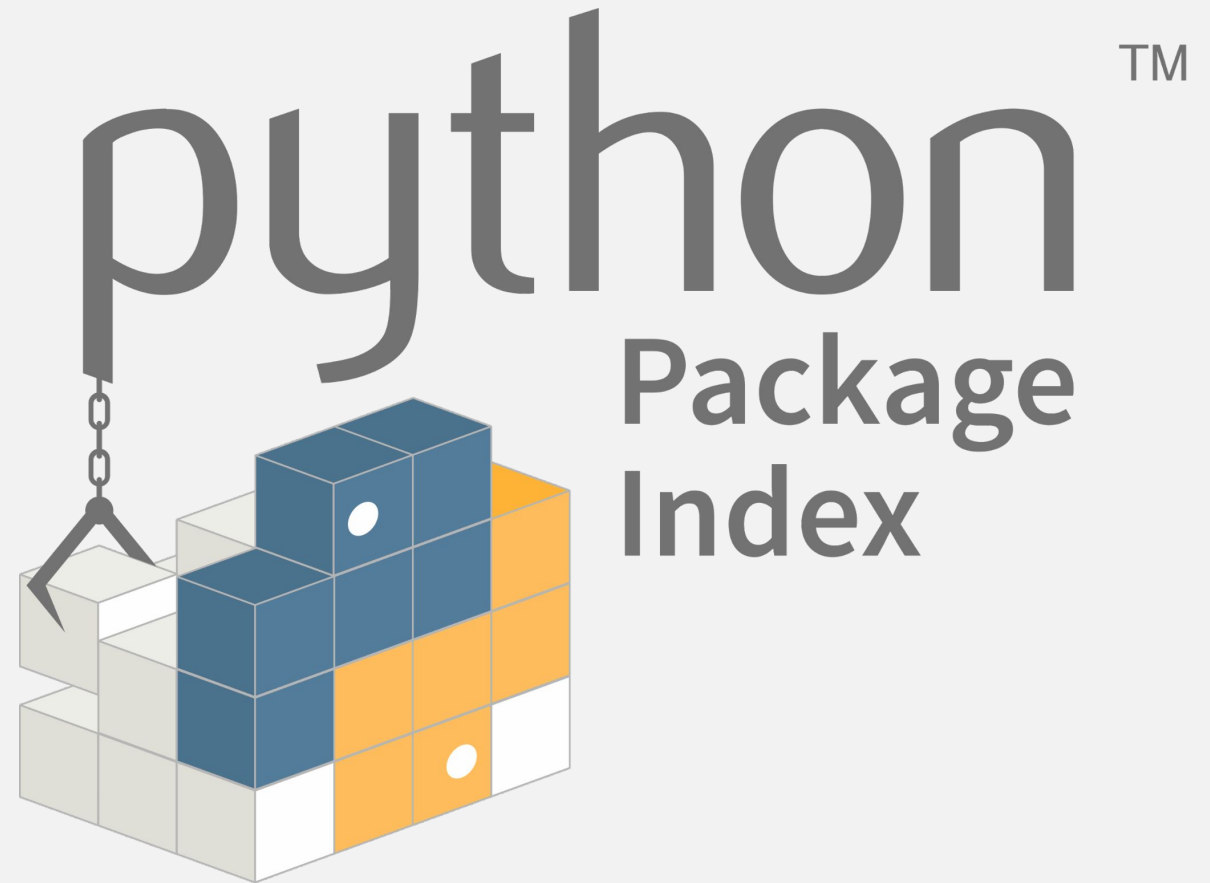
Fue creado en 2008 por Ian Bicking, como una herramienta para instalar y administrar paquetes de Python. Antes de PIP, el sistema de gestión era “*easy_install*”, pero PIP trajo varias mejoras.

1. Versión 1.x : Durante esta versión ganó popularidad pero con limitaciones.
2. Versión 6.x : Introdujo la resolución de dependencias.
3. Versión 7.x : Mejoras en velocidad y rendimiento
4. Versión 8.x : Soporte para la instalación de paquetes.
5. Versión 9.x : Mejoró la gestión de cache y nuevas características.
6. Versión 10.x : Legibilidad de los msj de errores y advertencias.
7. Versión 20.x : Una versión importante que introdujo resolución a las dependencias.
8. Versión 21.x : Mejoras en velocidad y la opcion use-future=in-tree-build.
9. Versión reciente: Sigue actualizando, mejorando su rendimiento.

¿Qué es PyPI?

Un repositorio en línea que almacena paquetes de software desarrollados para el lenguaje de programación Python.

Funciona como un almacén. Este repositorio tiene una amplia gama de paquetes que abarcan desde bibliotecas hasta frameworks para desarrollo web.





Se puede ingresar a su web a través de pypi.org

Cada paquete tiene una página donde encontrar información sobre la versión actual, documentación, y enlaces a su código fuente y repositorio.

Instalar

Para instalar un paquete de PyPI, puedes usar el comando `pip install` seguido del nombre del paquete que deseas instalar.

- *`pip install request`*
- *`py -m pip install "SomeProject"`*
- *`py -m pip install "SomeProject">=1,<2"`*
- *`py -m pip install "SomeProject"~=1.4.2"`*

```
C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>pip install camelcase
```

Actualizar

Para actualizar un paquete de PyPI hay dos maneras.

- *pip install "SomeProject"*
- *py -m pip install --upgrade "SomeProject"*

```
C:\Users\usuario>python -m pip install --upgrade pip
Collecting pip
  Downloading https://files.pythonhosted.org/packages/f9/fb/863012b13912709c13cf5cfdbfb304fa6c727659d6290438e1a88df9d848/pip-19.1-py2.py3-none-any.whl (1.4MB)
    100% |                               | 1.4MB 927kB/s
Installing collected packages: pip
  Found existing installation: pip 10.0.1
  Uninstalling pip-10.0.1:
    Successfully uninstalled pip-10.0.1
  Successfully installed pip-19.1
C:\Users\usuario>
```

Remover

Para eliminar un paquete instalado, puedes usar el comando `pip uninstall` seguido del nombre del paquete que deseas eliminar.

- *`pip uninstall "SomeProject"`*

```
C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>pip uninstall camelcase
```

```
Uninstalling camelcase-02.1:
```

```
Would remove:
```

```
c:\users\Your Name\appdata\local\programs\python\python36-32\lib\site-packages\camelcase-0.2-py3.6.egg-info
```

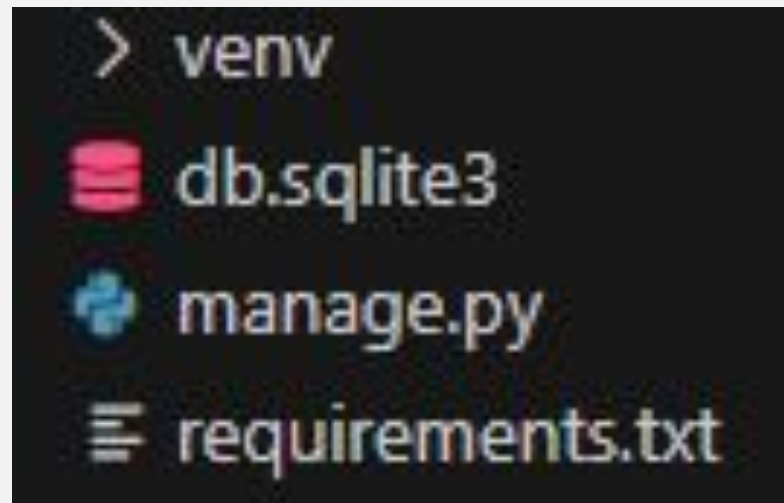
```
c:\users\Your Name\appdata\local\programs\python\python36-32\lib\site-packages\camelcase\*
```

```
Proceed (y/n)?
```


Requirements.txt

Con pip es posible crear un archivo de texto llamado requirements en el que se lista todos los paquetes que se hayan instalado en el proyecto.

- *python -m pip freeze > requirements.txt*



Requirements.txt

≡ requirements.txt

```
1  alembic==1.5.5
2  aniso8601==9.0.0
3  click==7.1.2
4  Flask==1.1.2
5  flask-marshmallow==0.14.0
6  Flask-Migrate==2.7.0
7  Flask-RESTful==0.3.8
8  Flask-SQLAlchemy==2.4.4
9  itsdangerous==1.1.0
10 Jinja2==2.11.3
11 Mako==1.1.4
12 MarkupSafe==1.1.1
13 marshmallow==3.10.0
14 marshmallow-sqlalchemy==0.24.2
15 python-dateutil==2.8.1
16 python-editor==1.0.4
17 pytz==2021.1
18 six==1.15.0
19 SQLAlchemy==1.3.23
20 Werkzeug==1.0.1
21
```

py -m pip install -r requirements.txt

Otros comandos importantes

pip list

Lista todos los paquetes instalados y sus versiones.

```
C:\> py -m pip list
Package Version
-----
docopt    0.6.2
idlex     1.13
jedi      0.9.0
```

pip show "SomeProject"

Proporciona detalles de un paquete específico.

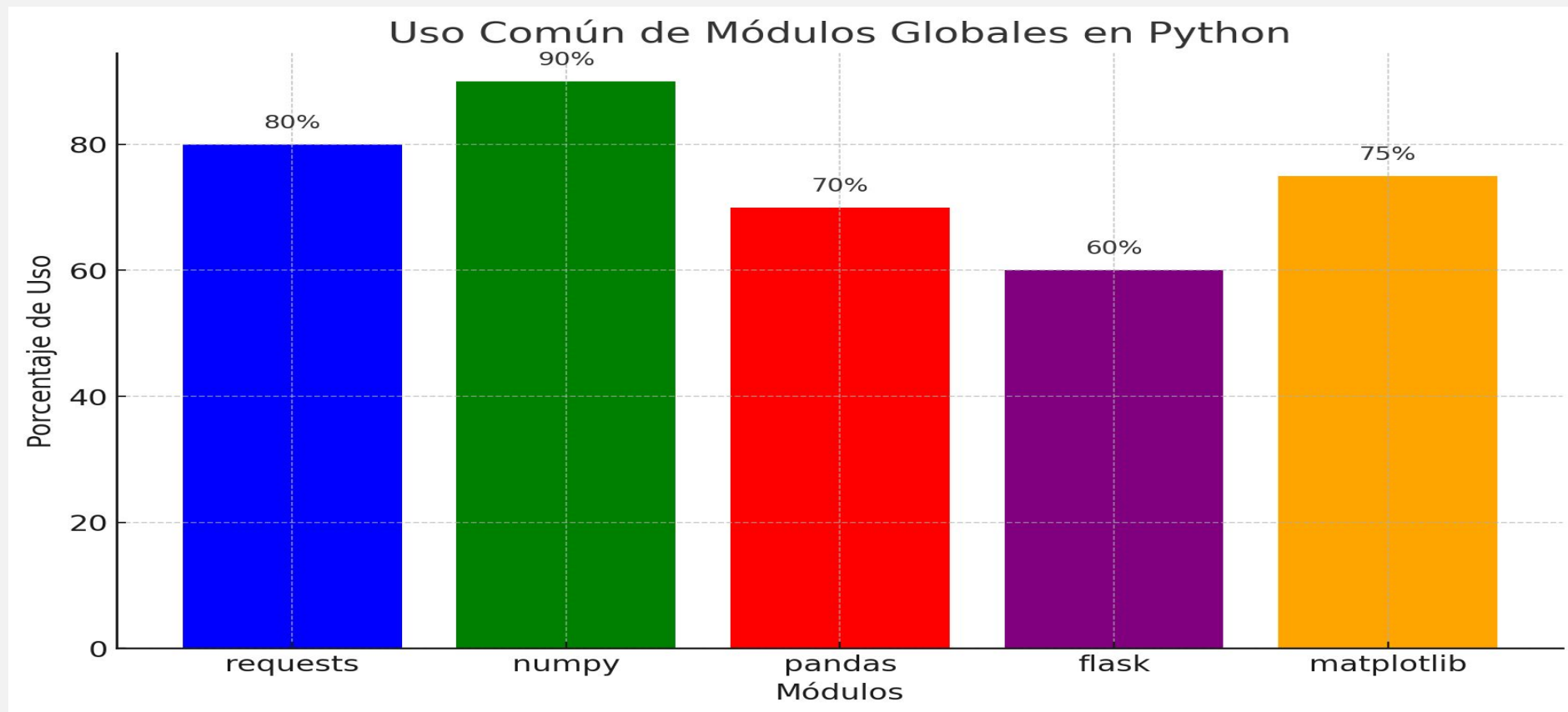
```
$ python -m pip show sphinx
Name: Sphinx
Version: 1.4.5
Summary: Python documentation generator
Home-page: http://sphinx-doc.org/
Author: Georg Brandl
Author-email: georg@python.org
License: BSD
Location: /my/env/lib/python2.7/site-packages
Requires: docutils, snowballstemmer, alabaster, Pygments, imagesize, Jinja2, babel, six
```

pip --help

Proporciona una guía de ayuda y una visión general de los comandos

¿Qué son los módulos globales?

Los módulos globales en Python son bibliotecas o paquetes instalados en un entorno de sistema que están disponibles para cualquier script o aplicación en dicho entorno.



Diferencias al instalar módulos

	GLOBAL	LOCAL
ALCANCE	Disponibles para todos los proyectos y scripts en el sistema.	Instalados en un entorno virtual específico y solo disponibles para proyectos dentro de ese entorno.
INSTALACIÓN	Instalados sin un entorno virtual o utilizando pip sin un prefijo de entorno virtual.	Instalados dentro de un entorno virtual utilizando herramientas como venv o virtualenv.
USO	Convenientes para herramientas y scripts que se utilizan a nivel de sistema.	Aislados y evitan conflictos de versiones entre diferentes proyectos.

¿Qué son los módulos propios?

Un módulo propio en Python es un archivo que creamos nosotros mismos, permitiendo su total personalización y edición. Esto facilita su mantenimiento y lo hace reutilizable en cualquier proyecto donde lo necesitemos.

 principal

```
✓ def saludar(nombre):  
    print(f"¡Hola, {nombre}!")  
  
✓ class Punto:  
    ✓ def __init__(self):  
        print("Esto es el init")
```

¿Cómo crear módulos propios y gestionarlos?

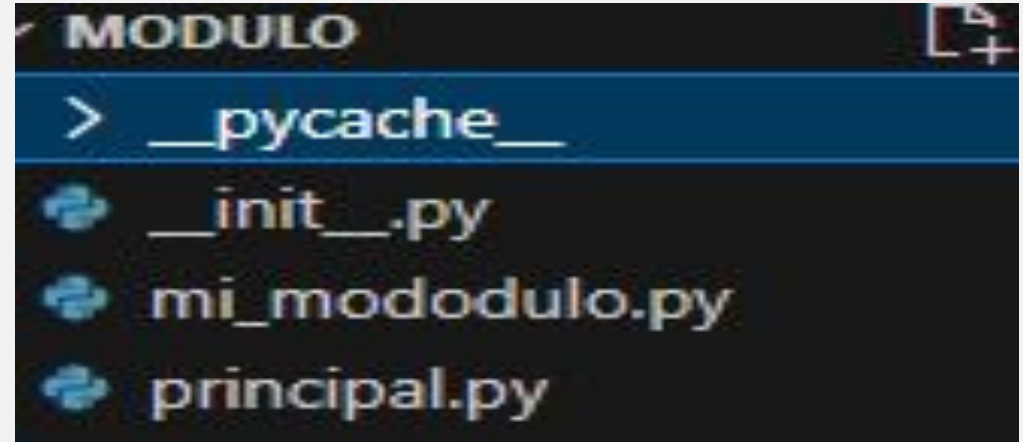
Crear un módulo es simple, se genera un archivo compuesto de código (funciones, clases, variables) con la extensión '.py'.

Para reutilizar el módulo en otros proyectos tendremos que utilizar la declaración "import MODULO" o "from PAQUETE import MODULO".

```
mi_mododulo.py •  
  
mi_mododulo.py > despedirse  
1 def saludar(nombre):  
2     return f" hola,{nombre}"  
3 def despedirse(nombre):  
4     return f"adios,{nombre}"  
5
```

```
mi_mododulo.py • principal.py X  
  
principal.py  
1 import mi_mododulo  
2  
3 print(mi_mododulo.saludar("carlos"))  
4 print(mi_mododulo.despedirse("carlos"))
```


Los módulos propios se gestionan en directorios usando el archivo '___init__.py' caracterizando los paquetes.



Alcance y Uso	Mantenimiento y Control	Dependencias
Proyecto específico: Parte del código del proyecto, no global.	Autonomía: El equipo controla implementación y cambios.	Internas: Dependen de módulos propios, no afectan a otros proyectos.
Reutilizables: Usables en otros proyectos si se decide.	Flexibles: Modificables sin terceros.	Locales: Gestión de versiones y actualizaciones dentro del proyecto.

Enlaces de interés

The History of PIP :

<https://www.nickmccullum.com/history-python-package-managers/#pip>

PIP en PyPI :



<https://pypi.org/project/pip/>

Wikipedia PIP :


[https://en.m.wikipedia.org/wiki/Pip_\(package_manager\)](https://en.m.wikipedia.org/wiki/Pip_(package_manager))

Documentación oficial PIP :

<https://pip.pypa.io/en/stable/getting-started/>



**¡Gracias
por su
atencion!**



- ♦ Walter Ibarrola
 - ♦ Rodrigo Trillo
 - ♦ Ivan Chirino
 - ♦ Luciano Morlio
 - ♦ Damián Libovich
- 