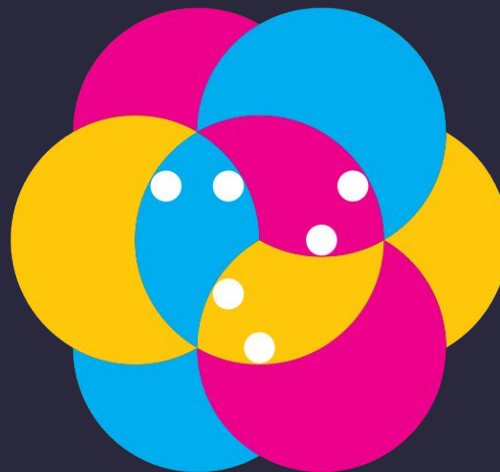




# PACKAGING EN PYTHON

Hay que dejar de pinear  
pip por al menos 2 años





# <Hello />

Soy Nahuel Ambrosini  
Me gusta programar  
Me gusta aprender  
> Me define la tensión entre mi vagancia <  
y mi obstinación





# /CONTENIDOS



## /01 /INTRO

- > Empaquetando mi proyecto

## /02 /DESAFÍOS

- > No es todo tan fácil

## /03 /TIPS

- > Pero todo tiene solución™

## /04 /DEMO

- > Veamos un ejemplo





# /01 - INTRO

Empaquetando mi primer proyecto





# /PRIMEROS PASOS



## /PIP FREEZE > REQS.TXT



```
Flask==2.0.2  
requests==2.7.0
```



```
# README.md  
$ pip install -r requirements.txt
```



## /CREATE SETUP.PY



```
from setuptools import setup  
  
with open('requirements.txt') as f:  
    reqs = f.read()  
  
setup(  
    ...  
    install_requires=reqs  
    ...  
)
```





# NO!



Por favor, no lo  
hagas!



Referencia: <https://caremad.io/posts/2013/07/setup-vs-requirement/>




INDEX.HTML



# /QUE TIENE DE MALO ESTO?



En principio nada..

-  Funciona
-  Es simple
-  Está documentado

A mitad de camino.. (Si el proyecto la rompe)

- Cuales son mis deps reales y cuales [sub]\*deps?
- Las debo pinear a todas? Y si sale alguna vuln de seguridad?
- Estoy instalando alguna que ya no use mas?
- Y si tengo una dependencia privada que no está en pypi como la instalo?

El final inexorable 

- **Conflictos de versiones!** Necesito ayuda, no me instala el proyecto *pero tiene que salir!*<sup>TM</sup>



# Ya sé!

> Pineo PIP a la versión <  
que andaba y nos re  
vimos  
—————→







# /02

# /DESAFÍOS

Cómo empaquetar una aplicación y sus dependencias





“Simple is better than complex.  
Complex is better than complicated”

— **Zen of Python**





# /DESAFÍOS DEL PACKAGING Y DEP MANAGEMENT

- Definir público objetivo (Desktop App, Mobile App, ...)
- Empaquetamos librerías y aplicaciones para ser pip instalables
- No ignorar la complejidad
- Manejar dependencias transitivas
- Crear entornos reproducibles
- Generar artefactos reproducibles
- Gestionar actualización de librerías vulnerables
- Manejar Compatibilidad con versiones de Python versions
- Instalar paquetes privados
- Hay que lidiar con paquetes con distintos estándares
- Actualizarse a los nuevos estándares del packaging





# /03

## /CONSEJOS

Cómo podemos resolver la complejidad del  
asunto





# /Consejos para humanos



1. Distinguir si el proyecto es una librería o una aplicación (Reutilizable por otros o no? Ese otro podes ser vos en otro proyecto también)
2. Si es una librería, debe ser laxo en las dependencias, estás para servir al prójimo, no para imponer versiones.
3. Si es una aplicación, debés garantizar la reproducibilidad al máximo
4. Tratar a las dependencias internas como si fueran una lib externa mas. Trackearlas en pip, con versionado y no instalarlas desde el código fuente. Evita problemas de ssh, seguimiento de cambios, versionado. Son libs, lo único especial es que las escribimos nosotros
5. Automatizar todo lo posible (Upgrade de deps, release versions, security checks, etc.)
6. Aprovechar las tools gratuitas (y financiarlas si podemos) para no reinventar lo creado.
7. Monitorear y adoptar los nuevos estándares.





# /Consejos para tools



- Distinguir entre mis dependencias, y las indirectas
- Facilitar configurar dependencias opcionales para testing/dev
- Facilitar la actualización de mis dependencias
- Permitir eliminar mis dependencias transitivamente si las dejo de usar
- Resolver a un conjunto de deps determinísticos a partir de las deps directas que declaro. (hashes for extra sec)
- Reproducir en un entorno aislado a partir de las deps determinísticas
- Adhesión a estándares (El ecosistema está en evolución)
- Buena Documentación, soporte y mantenibilidad





# /QUÉ TOOL ES MEJOR?



## /ÚNICO PROPÓSITO

- Build
- Twine
- PIPx
- venv



## /MULTI PROPÓSITO

- Poetry
- pip-tools
- pipenv
- hatch







# /04

# /DEMO

Veamos un ejemplo de una lib tirana



# /ONE TOOL TO RULE THEM ALL?

setup.py	Modern Way
setup.py sdist bdist_wheel	python -m build
setup.py install	pip install
setup.py develop	pip install -e .
setup.py upload	twine upload

# /CONCLUSIONES

- **Lucharás contra la opresión** de las librerías tiranas que imponen sus deseos como Ley.
- **No impondrás** como tirano tus propias versiones restrictivas, ni siquiera a vos mismo
- **No comprometerás** la mantenibilidad de tus dependencias a costo de la reproducibilidad. Para eso distinguir declaración de deps de archivo del reproducibilidad (lock file) puede ser muy útil
- **Aceptarás la complejidad** de manejar dependencias en un proyecto de gran escala y elegirás herramientas que lo faciliten, conociendo los casos de uso a resolver
- **Resistirás la tentación** de crear una solución superadora que no cumpla los estándares.
- **Automatizarás** todo lo posible para liberar al hombre de las tediosas tareas manuales
- **Utilizarás** el hash de las librerías instaladas para garantizar su integridad



# /REFERENCIAS



[Gist Link](#)





# <Gracias />



Preguntas?

