

Introducción

En este punto, has creado pequeños programas que utilizan las bibliotecas que vienen con la instalación de Python. Funcionan bien. Pero a medida que construyes programas más avanzados, debes hacerte una pregunta: ¿Cuánto de esta nueva funcionalidad escribo y cuánto ya está escrito que puedo usar en mis propios programas?

Como desarrollador, es probable que crees muchos programas en una sola máquina. Esos programas también se pueden usar en otras máquinas. Sin embargo, algunas de estas máquinas pueden tener versiones de Python instaladas que no son lo que espera. O bien, pueden tener bibliotecas instaladas que son de versión inferior o superior a las que necesita su programa.

Entonces, ¿Qué hacer? Debes encontrar una manera de que tu programa funcione de forma aislada, para que no moleste lo que está instalado en la máquina de destino. También debes asegurarte de que una máquina no deshabilite tu programa porque tiene instalada la versión incorrecta de Python o la versión de la biblioteca.

Escenario: Construyamos un programa

A medida que tus habilidades crecen y comienzas a construir programas más avanzados, siempre deseas comenzar con un buen enfoque. Deseas pensar en el programa que compila como un proyecto, con el código distribuido en muchos archivos. Si es posible, también deseas utilizar bibliotecas que otros han escrito, para acelerar el tiempo de desarrollo.

¿Qué aprenderás?

Al final de este módulo, podrás:

- Crear un proyecto de Python.

- Desarrollar y ejecutar tu código de forma aislada en una máquina.
- Usar bibliotecas que otra persona haya escrito.
- Restaurar un proyecto a partir de una lista de dependencias.

¿Cuál es el objetivo principal?

Utilizar bibliotecas y planificar tu proyecto para crear programas Python que sean más avanzados.

Trabajar con paquetes

La mayoría de los programas que escribas se basarán en código escrito por otros. Este código a menudo viene en forma de paquetes, que son módulos externos o bibliotecas que se incluyen en el proyecto. Al igual que con cualquier proyecto que requiera un conjunto de recursos, es importante considerar cómo te asegurarás de que los recursos adecuados estén disponibles para tu programa.

Un buen comienzo es aprender a administrar tu programa. Una forma de hacerlo es pensar en el programa como un proyecto. Python aborda esto mediante el uso de algo llamado entornos virtuales.

¿Qué es un entorno virtual?

Tienes una máquina de desarrollo. En esa máquina, es posible que tengas una versión de Python instalada o una versión de una biblioteca que quieras usar. ¿Qué sucede cuando mueves tu programa a una máquina que tiene una versión diferente de Python instalada o diferentes versiones de las bibliotecas de las que depende?

Una cosa que no deseas hacer es asumir que tu programa funcionará y que puede instalar la última versión de las bibliotecas de las que depende. Si haces eso, podrías terminar destruyendo la capacidad de los otros programas para funcionar en la máquina de destino. La solución es encontrar una manera de que tu aplicación funcione de forma aislada.

La solución de Python para estos problemas es un entorno virtual. Un entorno virtual es una copia autónoma de todo lo necesario para ejecutar el programa. Esto incluye el intérprete de Python y cualquier biblioteca que su programa necesite. Mediante el uso de un entorno virtual, puedes asegurarte de que tu programa tendrá acceso a las versiones y recursos correctos para ejecutarse correctamente.

El flujo de trabajo básico se ve así:

- Crear un entorno virtual que no afecte al resto de la máquina.
- Ingresar al entorno virtual, donde especifique la versión de Python y las bibliotecas que necesita.
- Desarrolla tu programa.

Crear un entorno virtual

Para crear un entorno virtual, llama al módulo. El módulo espera un nombre como argumento `venv`.

Sigue estos pasos:

Desde tu terminal ve al directorio donde desees guardar tu proyecto.

(Documentos/TuFolderPreferido) Ejemplo en windows: `cd Documents/TuFolderPreferido`

Utiliza el siguiente comando para llamar al módulo `venv`. El comando difiere ligeramente dependiendo de tu sistema operativo.

En consola: `python3 -m venv env`

```
~/Documents/CursoIntroPython-main/Módulo 2 - Crear y administrar proyectos  
> python3 -m venv env
```

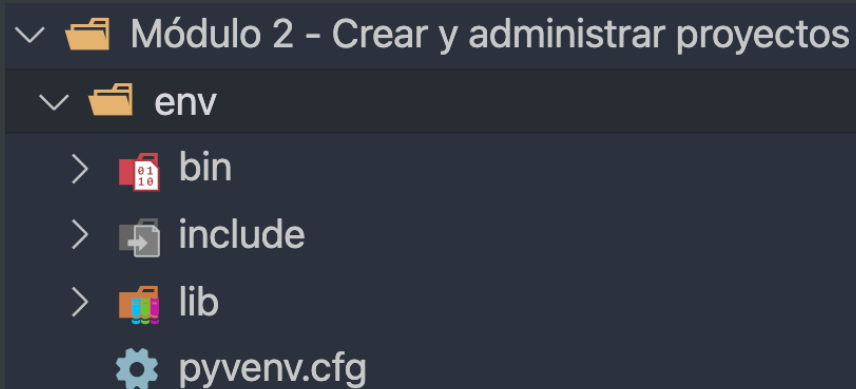
Donde:

- `python3`: La versión de python a utilizar.

- `venv`: Llamada al módulo `venv` conocido como virtual environment.
- `env`: Nombre de nuestro entorno virtual.

En este punto, se crean algunos directorios:

```
/env
  /bin
  /include
  lib
```






Tu entorno necesita el directorio `venv` para realizar un seguimiento de detalles como qué versión de Python y qué bibliotecas está utilizando. No coloque los archivos de programa en el directorio `venv`. Te sugerimos que coloques tus archivos en el directorio `src` o algo similar. La estructura del proyecto podría verse así:

```
/env
/src
  program.py
```

Creando la carpeta `/src` mediante la terminal usando el comando `mkdir`

```
~/Documents/CursoIntroPython-main/Módulo 2 - Crear y administrar proyectos
> mkdir src

~/Documents/CursoIntroPython-main/Módulo 2 - Crear y administrar proyectos
> █
```

```
✓  Módulo 2 - Crear y administrar proyectos
>  env
>  src
```

Activar el entorno virtual

En este punto, tienes un entorno virtual, pero no has comenzado a usarlo. Para usarlo, debes activarlo llamando `activate` a un script en tu directorio `env`.

Así es como puede verse la activación en distintos sistemas operativos.

```
# Bash | Consola
# Windows
env\bin\activate

# Linux, WSL o macOS
source env/bin/activate
```

Llamar al comando `activate` cambia el mensaje de salida de la consola. Ahora está precedido con `(env)` y se parece a este ejemplo:

```
# Bash | Consola
(env) -> path/to/project
```

```
~/Documents/CursoIntroPython-main/Módulo 2 - Crear y administrar proyectos
> source env/bin/activate

~/Documents/CursoIntroPython-main/Módulo 2 - Crear y administrar proyectos
env > █
```

Ahora, ya estás dentro de tu entorno virtual. Cualquier cosa que hagas sucede de forma aislada.

¿Qué es un paquete?

Una de las principales ventajas de utilizar bibliotecas externas es acelerar el tiempo de desarrollo de tu programa. Puedes obtener una biblioteca de este tipo en Internet. Pero al buscar e instalar estas bibliotecas a través de un entorno virtual, te aseguras de instalar estas bibliotecas solo para el entorno virtual y no globalmente para toda la máquina.

Instalar un paquete

Instalar un paquete mediante `pip`. El comando `pip` utiliza el Python Package Index, o PyPi para abreviar, para saber dónde obtener los paquetes. Puedes visitar el sitio web de [PyPi](#) para conocer qué paquetes están disponibles.

Para instalar un paquete, ejecute , como en este ejemplo: `pip install`

Dato curioso: Si estás desde un notebook se ejecuta así: `!pip install python-dateutil`

Con signo de admiración al inicio. Sin embargo, no estamos trabajando con notebooks ahorita, estamos ejecutando todo por terminal (consola, bash, cli, cmd, como sea que le digas).

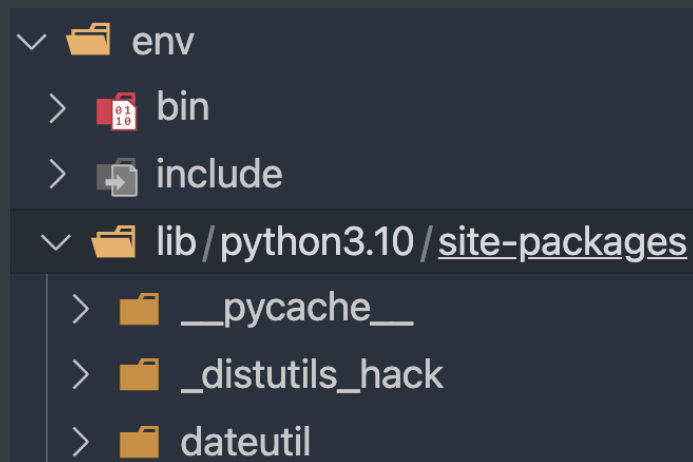
```
# Bash | Consola
pip install python-dateutil
```

Si ejecutas el comando anterior, descargará e instalará `dateutil`, un paquete para analizar el formato de archivo `.yaml`. Después de instalar el paquete, puedes verlo en la lista si expande el directorio `lib` en `env`, así:

```
# Mensaje de salida en consola
/env
/lib
/dateutil
```

```
~/Documents/CursoIntroPython-main/Módulo 2 - Crear y administrar proyectos
env > pip install python-dateutil
Requirement already satisfied: python-dateutil in ./env/lib/python3.10/site-packages (2.8.2)
Requirement already satisfied: six>=1.5 in ./env/lib/python3.10/site-packages (from python-dateutil) (1.16.0)

~/Documents/CursoIntroPython-main/Módulo 2 - Crear y administrar proyectos
env > █
```



Para ver qué paquetes están ahora instalados en tu entorno virtual, puedes ejecutar `pip freeze`. Este comando produce una lista de paquetes instalados en el terminal:

Recuerda: Si deseas instalar un paquete desde un notebook se ejecuta con signo de admiración al inicio. `!pip freeze` Sin embargo, no estamos trabajando con notebooks ahorita, estamos ejecutando todo por terminal (consola, bash, cli, cmd, como sea que le digas).

```
# Mensaje de salida en consola
python-dateutil==2.8.2
six==1.16.0
```

Contiene algo más que sólo pipdate por que en sí misma se basan otras bibliotecas.

```
~/Documents/CursoIntroPython-main/Módulo 2 - Crear y administrar proyectos
env > pip freeze
python-dateutil=2.8.2
six=1.16.0
```

Para asegurarte de que estos paquetes solo existen en tu entorno virtual, intenta salir de ese entorno llamando al comando `deactivate` :

```
# Bash | Consola
deactivate
```

Observa cómo cambia el mensaje de la terminal. Ya no está precedido por `(env)` y ha regresado a su estado anterior:

```
# Bash | Consola
path/to/project
```

```
~/Documents/CursoIntroPython-main/Módulo 2 - Crear y administrar proyectos
env > deactivate

~/Documents/CursoIntroPython-main/Módulo 2 - Crear y administrar proyectos
> █
```

Si ejecutas el comando `pip freeze` , verás una lista mucho más larga de dependencias. Esta lista indica que verás todos los paquetes instalados en tu máquina en lugar de solo lo que está instalado en tu entorno virtual.


```
> pip freeze
appnope==0.1.2
argon2-cffi==21.3.0
argon2-cffi-bindings==21.2.0
asttokens==2.0.5
attrs==21.4.0
backcall==0.2.0
black==22.1.0
bleach==4.1.0
cffi==1.15.0
click==8.0.3
cyclr==0.11.0
debugpy==1.5.1
decorator==5.1.1
defusedxml==0.7.1
entrypoints==0.4
executing==0.8.2
fonttools==4.29.1
ipykernel==6.8.0
ipython==8.0.1
ipython-genutils==0.2.0
ipywidgets==7.6.5
jedi==0.18.1
Jinja2==3.0.3
jsonschema==4.4.0
jupyter-client==7.1.2
jupyter-core==4.9.1
jupyterlab-pygments==0.1.2
jupyterlab-widgets==1.0.2
kiwisolver==1.3.2
MarkupSafe==2.0.1
matplotlib==3.5.1
matplotlib-inline==0.1.3
mistune==0.8.4
mypy-extensions==0.4.3
nbclient==0.5.10
nbconvert==6.4.1
nbformat==5.1.3
nest-asyncio==1.5.4
notebook==6.4.8
numpy==1.22.2
packaging==21.3
pandocfilters==1.5.0
parso==0.8.3
pathspec==0.9.0
pexpect==4.8.0
pickleshare==0.7.5
Pillow==9.0.1
platformdirs==2.4.1
prometheus-client==0.13.1
prompt-toolkit==3.0.26
ptyprocess==0.7.0
pure-eval==0.2.2
pycparser==2.21
Pygments==2.11.2
pyparsing==3.0.7
pysistent==0.18.1
python-dateutil==2.8.2
pyzmq==22.3.0
Send2Trash==1.8.0
six==1.16.0
stack-data==0.1.4
terminado==0.13.1
testpath==0.5.0
tomli==2.0.0
tornado==6.1
```

Más formas de instalar un paquete

También puedes utilizar los siguientes comandos para instalar un paquete:

- Teniendo un conjunto de archivos en tu máquina e instalándolos desde esa fuente:

```
# Bash | Consola
cd <to where the package is on your machine>
python3 -m pip install .
```

- Instalar desde un repositorio de GitHub que nos proporciona el control de versiones:

```
git+https://github.com/your-repo.git
```

- Instalar desde un archivo comprimido:

```
python3 -m pip install package.tar.gz
```

Usar un paquete instalado

Ahora tienes un paquete instalado. ¿Cómo se usa en el código?

Asegúrate de tener un directorio para tus archivos. Te sugerimos que llames al directorio (folder) src y agregues un archivo Python llamado app.py. Ahora agrega un poco de código para llamar al comando `pipdate` :

```
from datetime import *
from dateutil.relativedelta import *
now = datetime.now()
print(now)

now = now + relativedelta(months=1, weeks=1, hour=10)

print(now)
```

app.py



CursoIntroPython-main > Módulo 2 - Crear y administrar proyectos > src > app.py > ...

```
1 from datetime import datetime
2 from datetime import *
3 from dateutil.relativedelta import *
4 now = datetime.now()
5 print(now)
6
7 now = now + relativedelta(months=1, weeks=1, hour=10)
8
9 print(now)
```

```
~/Documents/CursoIntroPython-main/Módulo 2 - Crear y administrar proyectos/src
> python3 app.py
2022-02-08 02:04:28.025424
2022-03-15 10:04:28.025424
```

```
~/Documents/CursoIntroPython-main/Módulo 2 - Crear y administrar proyectos/src
> 
```

Curso Propedúctico de Python para Launch X - Innovación Virtual.

Material desarrollado con base en los contenidos de MSLearn y la metáfora de LaunchX, traducción e implementación por: Fernanda Ochoa - Learning Producer de LaunchX.

Redes:

- GitHub: [FernandaOchoa](#)
- Twitter: [@imonsh](#)
- Instagram: [fherz8a](#)