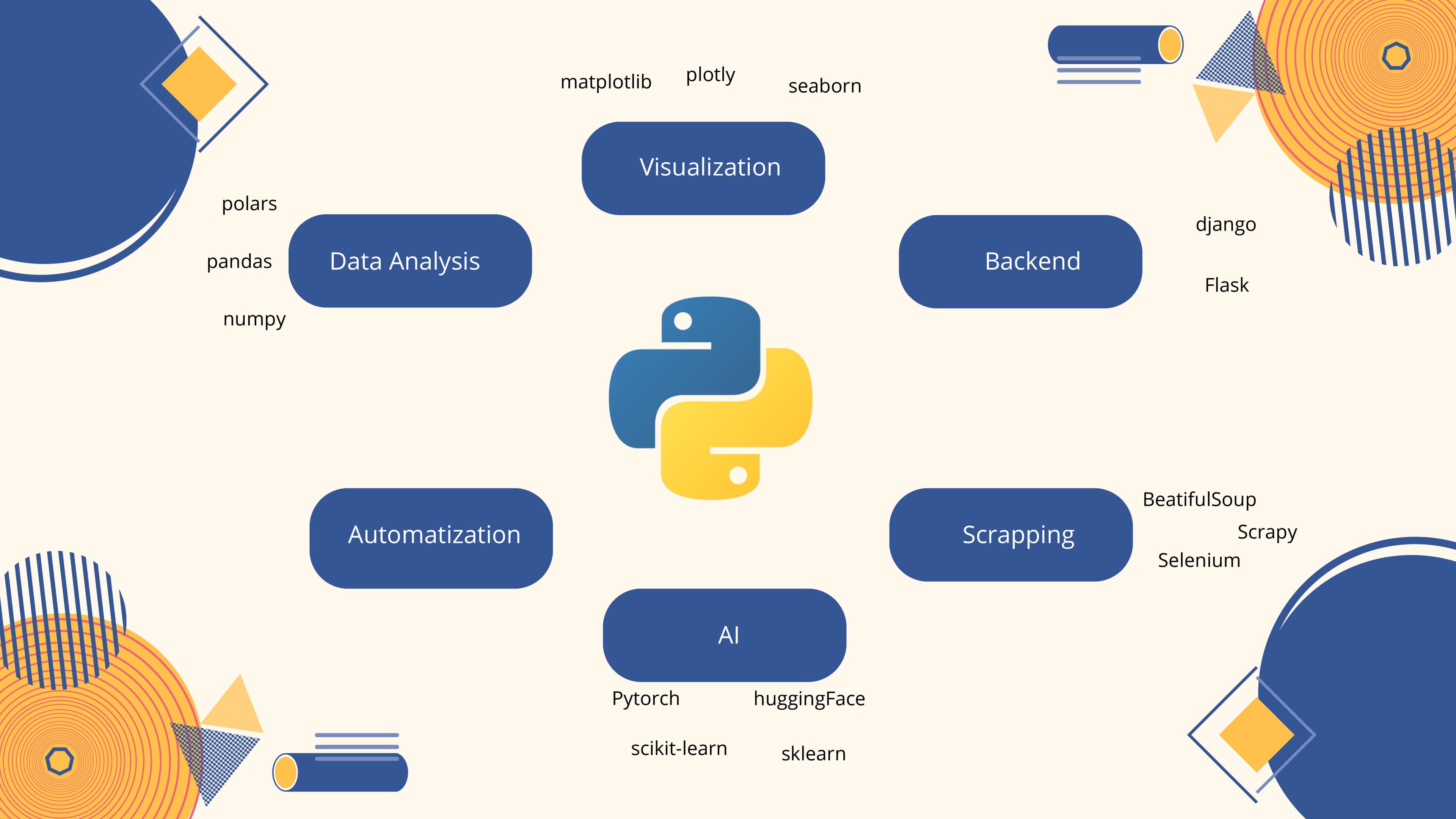


GINYARD UNIVERSITY

INTRO A PYTHON



QUE SE NECESITA PARA PROGRAMAR



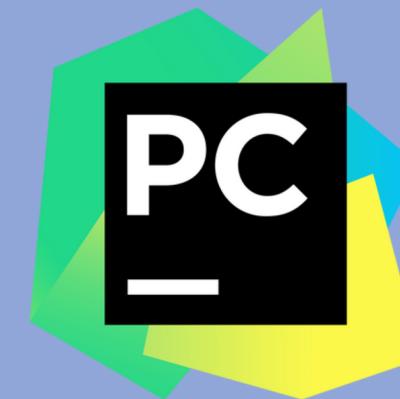
google colab
es una plataforma gratuita de Google que permite la ejecución de código Python en la nube, facilitando el desarrollo y la colaboración en proyectos de ciencia de datos y aprendizaje automático. Una ventaja clave es que no requiere configuración local y ofrece acceso a recursos de computación de alto rendimiento, como GPU y TPU, lo que acelera el procesamiento de grandes volúmenes de datos.



cursor
¿Qué es? Un editor de código basado en VSCode, pero con inteligencia artificial integrada para ayudarte a programar.
Ventajas clave: Su IA puede generar y explicar código, acelerando el flujo de trabajo. La interfaz es familiar si ya usas VSCode.
Desventajas clave: Aún está en desarrollo y su rendimiento depende de la calidad de las respuestas de la IA.



vscode
¿Qué es? Es un editor de código gratuito, ligero y muy popular de Microsoft.
Ventajas clave: Es muy rápido, se puede personalizar con miles de extensiones y funciona en cualquier sistema operativo.
Desventajas clave: Depende de extensiones para muchas funciones y puede tener una curva de aprendizaje inicial.



pycharm
¿Qué es? Un entorno de desarrollo (IDE) completo y potente, diseñado específicamente para el lenguaje Python.
Ventajas clave: Incluye herramientas avanzadas para depurar, probar y refactorizar código. Es ideal para proyectos grandes y complejos.
Desventajas clave: Es más pesado y consume más recursos que otros editores. La versión profesional es de pago.

**si programo código en
pycharm y luego paso mi
código a Vscode, tengo que
hacer algún cambio?**

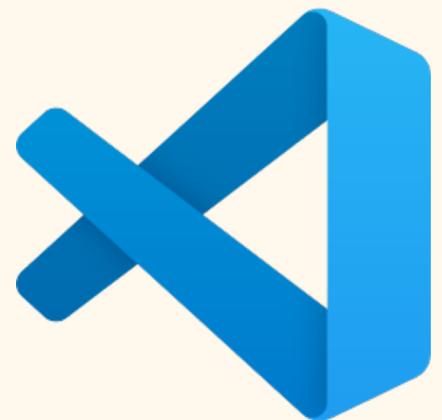
Si

No

Enviar

● Loading...

Instalemos VsCode



vscode

Extensiones utiles

python

jupyter

copilot

pylance

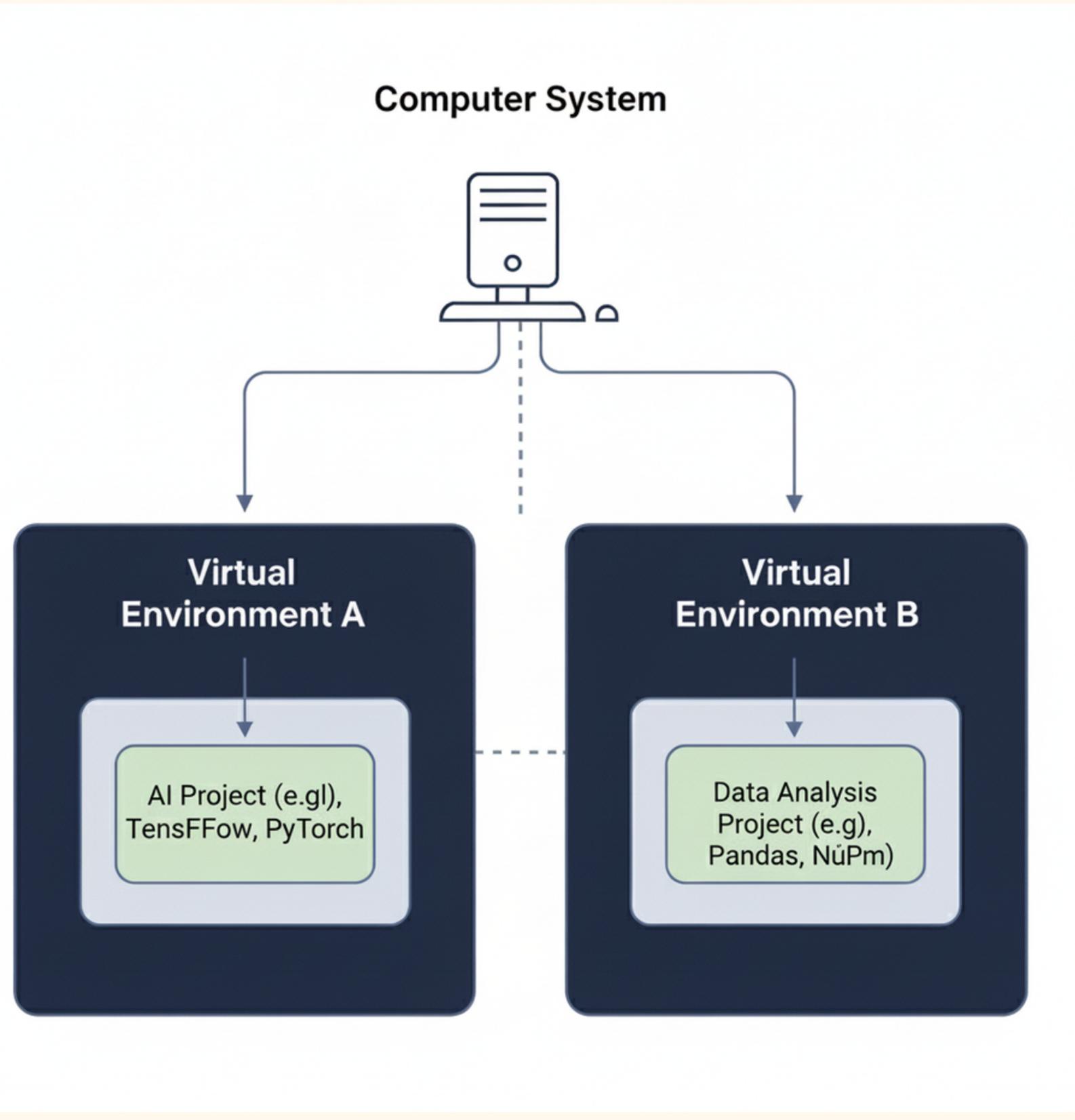
Icons

Themes

¿Qué es un Ambiente Virtual?

Imagina que estás construyendo un proyecto con piezas de LEGO. En lugar de mezclar todas las piezas de todos tus proyectos, un ambiente virtual te permite crear una caja separada, exclusiva para las piezas que necesitas para tu proyecto actual.

En la programación, un ambiente virtual es exactamente eso: un **espacio aislado** que **contiene todas las herramientas, librerías y dependencias** necesarias para un proyecto específico.



Pero me da lata tener que hacer un ambiente para cada proyecto, es posible tener solo uno y que todos mis proyectos hablen con ese?

Si

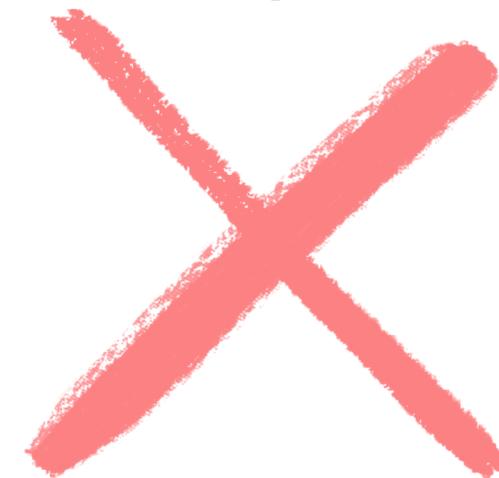
No

Enviar

● Loading...

Solo 1 ambiente

proyecto 1: AI —— ocupa —→ pytorch 2.0.4 — depende de —→ Numpy 2.0.1



errores

proyecto 2: Análisis —— ocupa —→ Pandas 2.0.1 — depende de —→ Numpy 1.2.6

ambiente 1

proyecto 1: AI → pytorch 2.0.4 → Numpy 2.0.1

ambiente 2

proyecto 2: Análisis → Pandas 2.0.1 → Numpy 1.2.6

Como manejamos un ambiente virtual

1. Pip: El Instalador Clásico

Pip es el gestor de paquetes que viene por defecto con Python. Es simple y directo.

- Función principal: Instalar, actualizar y desinstalar paquetes.
- Cómo funciona: Lee una lista de paquetes de un archivo requirements.txt.
- Su gran debilidad: No resuelve conflictos complejos entre dependencias. Si el Proyecto A necesita la librería X v1.0 y el Proyecto B necesita la librería X v2.0, pip no puede manejarlo bien, llevando al famoso "infierno de las dependencias".

En resumen: pip es como una lista de compras; te dice qué necesitas, pero no te asegura que todos los ingredientes funcionen bien juntos.



2. Poetry: El Gestor de Proyectos Todo-en-Uno

Poetry es una herramienta completa para gestionar todo el ciclo de vida de un proyecto.

- Función principal: Es mucho más que un instalador. Gestiona dependencias, ambientes virtuales y el empaquetado del proyecto para su publicación.
- Cómo funciona: Usa un archivo `pyproject.toml` para definir las dependencias y crea un `poetry.lock` que "congela" las versiones exactas que funcionan.
- Su gran fortaleza: Garantiza builds reproducibles. Todo el equipo de desarrollo trabajará con exactamente las mismas versiones de librerías, eliminando los conflictos.
- En resumen: Poetry es como la receta de un chef profesional; no solo lista los ingredientes, sino que te da las versiones y cantidades exactas que garantizan que el plato final sea perfecto, siempre.



3. Uv: El Futuro de la Velocidad

- Uv es la herramienta más nueva y su objetivo es uno solo: ser increíblemente rápida.
Función principal: Es un instalador y resolutor de dependencias ultrarrápido, escrito en Rust.
- Cómo funciona: Puede usarse como un reemplazo directo de pip. Es capaz de leer requirements.txt y pyproject.toml a una velocidad de 10 a 100 veces superior.
- Su lugar en el ecosistema: No es un gestor de proyectos como Poetry, sino un componente de alto rendimiento. De hecho, las versiones más recientes de Poetry ya permiten usar uv como su motor interno para acelerar drásticamente las instalaciones.

Quiero instalar Numpy —— ocupo —— **Manejador de ambiente (pip, poetry uv)**

Se instala en

carpeta .venv

donde tenemos
todas nuestra dependencias

MI proyecto habla con las dependencias

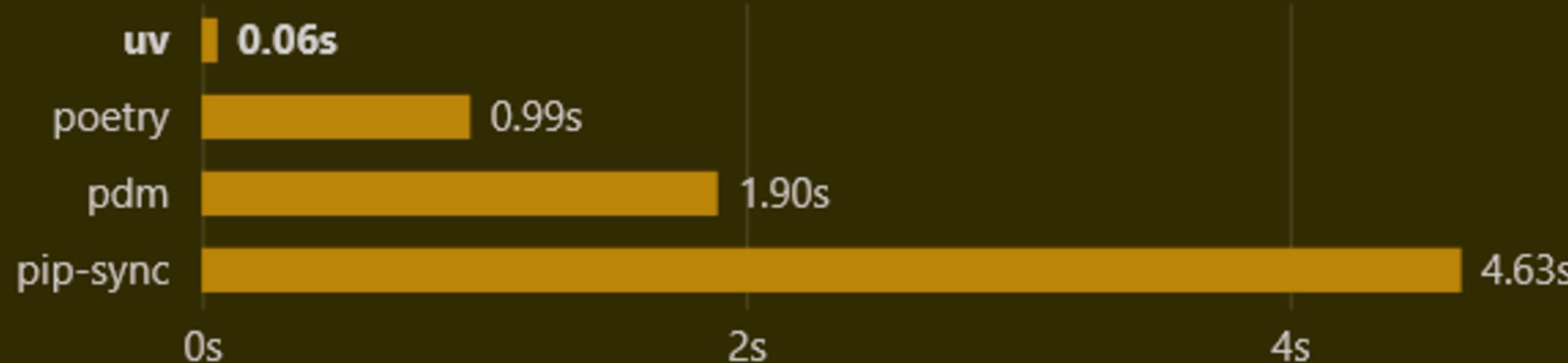
Proyecto A

UV

te permite manejar la version de python, dependencias etc.

UV

An extremely fast Python package and project manager, written in Rust.



*Installing **Trio**'s dependencies with a warm cache.*

Gestionar Versiones de Python

uv también puede instalar y gestionar las versiones de Python en tu sistema, actuando de forma similar a herramientas como pyenv.

- **uv python install <versión>**
- Descarga e instala una o más versiones específicas de Python. Por ejemplo, uv python install 3.12 instalará la versión más reciente de Python 3.12.
- **uv venv --python <versión>**
- Crea un ambiente virtual utilizando una versión específica de Python que ya tengas disponible. Por ejemplo, uv venv -p 3.11 (o --python 3.11) creará un ambiente que use Python 3.11, aunque tu versión global sea otra.
- **uv python pin <versión>**
- Fija una versión de Python para el directorio actual. Esto crea un archivo .python-version para que uv sepa qué versión de Python usar por defecto en ese proyecto.

Estos comandos te permiten manejar no solo las librerías de un proyecto, sino también la versión exacta del intérprete de Python con el que se ejecuta, todo desde la misma herramienta.

Crear y Activar un Ambiente Virtual



Estos comandos reemplazan a `python -m venv`.

- **uv venv** Crea un ambiente virtual en el directorio actual, dentro de una carpeta llamada `.venv`.
- **uv venv mi_entorno** Crea un ambiente virtual con un nombre personalizado.

Comandos Básicos de uv add

- **uv add <paquete>** Añade una dependencia de producción. Por ejemplo, uv add requests.
- **uv add <paquete> --dev** Añade una dependencia de desarrollo (para pruebas, linters, etc.). Por ejemplo, uv add pytest --dev.
- **uv remove <paquete>** Elimina una dependencia del pyproject.toml y la desinstala del ambiente.

Resumen Rápido: De Cero a "Hola Mundo" con uv

1. Instalar y Elegir la Versión de Python 🐍

Bash

```
# Instala la versión de Python que necesitas
```

```
uv python install 3.12
```

```
# Fija esa versión para tu proyecto actual
```

```
uv python pin 3.12
```

2. Crear y Activar el Ambiente Virtual 📦

Bash

```
# Crea el ambiente virtual (usará la versión que fijaste)
```

```
uv venv
```

3. Inicializar el Proyecto y Añadir Paquetes 🔧

Bash

```
# Crea el archivo pyproject.toml para tu proyecto
```

```
uv init
```

```
# Añade una dependencia (la instala y la guarda en pyproject.toml)
```

```
uv add "requests"
```

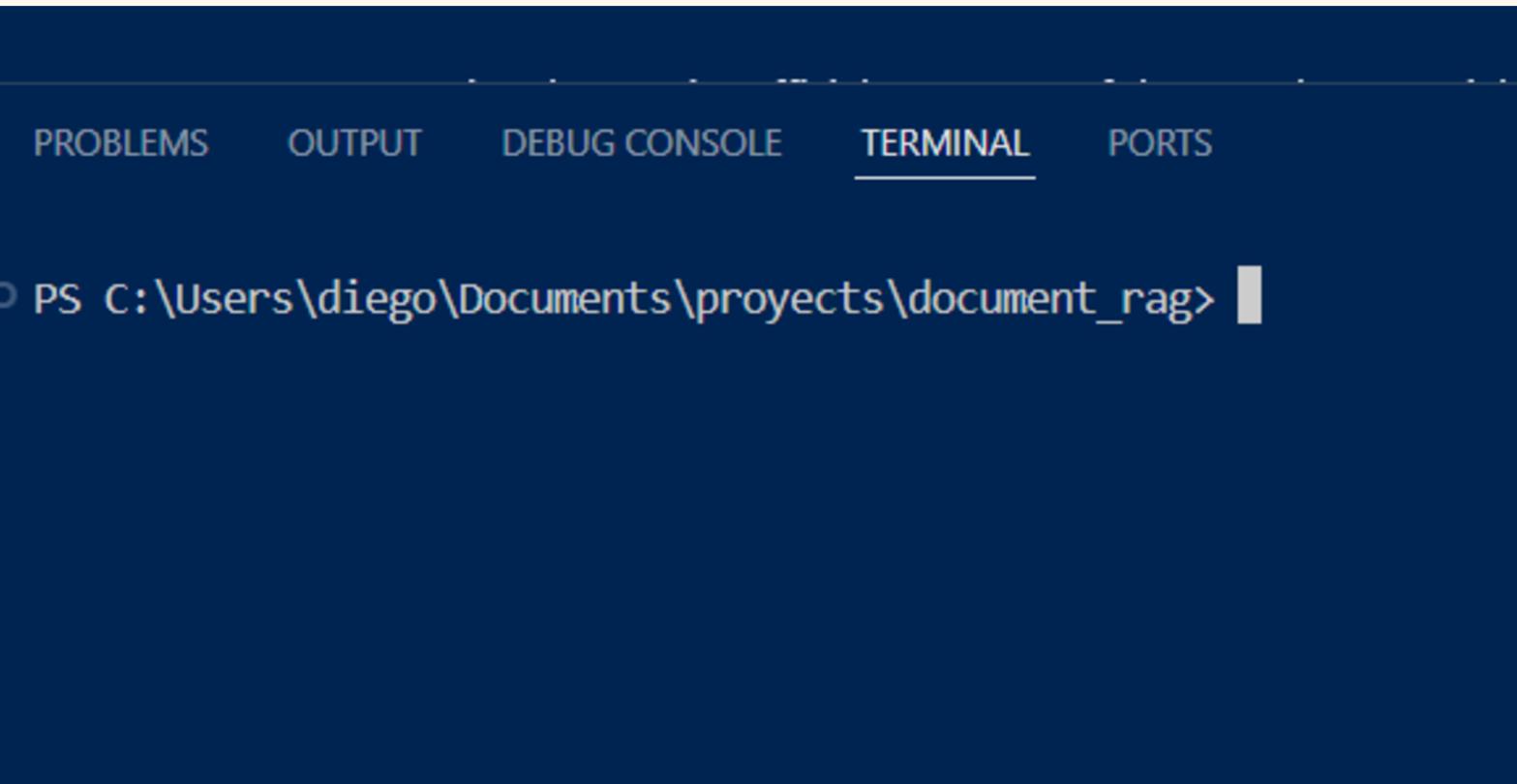
4. Ejecutar tu Script 🚀

Bash

```
# Corre tu script de Python dentro del ambiente
```

```
uv run python tu_script.py
```

los comandos se escriben aca



 **Editor de código**

 **Manejo de ambiente**

 **Entender los distintos tipos de archivos**



Entender los distintos tipos de archivos

.py

archivo script

.ipynb

cuaderno notebook
mismo que genera
google colab

.json

archivo que guarda
forma diccionario

.CSV

.xslx

tutorial de
programación