Entorno virtual en Python

> Para que sirve el entorno virtual en Python?

Un **entorno virtual en Python** (virtualenv o venv) sirve para **aislar los paquetes y dependencias de cada proyecto**, evitando conflictos entre ellos y con el sistema.

En otras palabras:
cada proyecto tiene su **propio Python y sus propias librerías**, sin afectar a los demás.

🔴 El problema sin entornos virtuales

Imagina que tienes dos proyectos:

- Proyecto A necesita: `Django 2.2`
- Proyecto B necesita: `Django 4.0`

Si instalas ambos globalmente:

```
bash

pip install django
```

Solo puede haber **una versión**, y uno de los proyectos se rompe 😖

🟢 La solución: Entornos virtuales

Con un entorno virtual:

```
Proyecto_A/
   venv/
   django 2.2

Proyecto_B/
   venv/
   django 4.0
```

Cada proyecto vive en su propio “mini-Python”.

🧠 ¿Para qué sirve en la práctica?

Sirve para:

Beneficio	Explicación
Evitar conflictos	Cada proyecto usa versiones diferentes

Beneficio	Explicación
Mantener orden	No llenas tu Python global de librerías
Reproducir proyectos	Otros pueden instalar exactamente lo mismo
Usar versiones viejas	Sin romper programas nuevos
Trabajar en equipo	Todos usan las mismas dependencias

🔪 Ejemplo real

Supón que haces un analizador semántico en Python (como el que querías para VS Code 👀).
Tal vez usas:

- `ply`
- `lark`
- `antlr4`
- `colorama`

No quieres que eso afecte otros proyectos como juegos, bots, etc.

Con un entorno virtual:

```
bash

python -m venv venv
```

Lo activas:

Windows

```
bash

venv\Scripts\activate
```

Linux/Mac

```
bash

source venv/bin/activate
```

Luego:

```
bash

pip install lark ply
```

Eso **solo se instala dentro del proyecto**.

🗒️ ¿Qué pasa si borras el entorno?

Nada grave 😎
Solo pierdes las librerías, pero puedes recrearlo con:

```
bash

pip install -r requirements.txt
```

🧠 En resumen

Un entorno virtual es como una **caja cerrada** para cada proyecto Python:

> "Aquí solo viven las librerías que este proyecto necesita".

Si quieres, luego te enseño cómo usar entornos virtuales dentro de VS Code paso a paso 🔥

> Que es una maquina virtual de python?

Una **máquina virtual de Python** es el "motor interno" que ejecuta los programas Python.
No es una computadora falsa como VirtualBox, sino un **intérprete que entiende el código Python**.

Se llama oficialmente:

> 🧠 **Python Virtual Machine (PVM)**

🧩 ¿Qué hace exactamente?

Cuando escribes esto:

```
python

print("Hola")
```

Python NO lo ejecuta directamente. Pasa por estas etapas:

1. **Compila** tu código `.py`
2. Lo convierte en **bytecode**
3. Ese bytecode lo ejecuta la **máquina virtual de Python**

```
scss

codigo.py
   ↓
Bytecode (.pyc)
   ↓
Python Virtual Machine (PVM)
```

```
   ↓
CPU real
```

🧠 ¿Qué es el bytecode?

Es una versión intermedia del código, por ejemplo:

```
arduino


LOAD_NAME print
LOAD_CONST "Hola"
CALL_FUNCTION
```

No es código máquina, es un lenguaje que solo entiende Python.

🧠 Entonces...

La **máquina virtual de Python** es un programa que:

- Lee bytecode
- Ejecuta instrucciones
- Maneja memoria, variables, objetos, funciones, etc.

Es lo que hace que Python sea:

- Portátil
- Independiente del sistema operativo

🟦 Importante: NO confundir

Entorno virtual (venv)	Máquina virtual (PVM)
Aísla librerías	Ejecuta el código
Es para proyectos	Es parte del intérprete
Es una carpeta	Es un motor
Se crea con **venv**	Viene dentro de Python

🧠 Ejemplo sencillo

Cuando corres:

```
bash


python mi_programa.py
```

Lo que realmente pasa es:

> La **máquina virtual de Python** lee el bytecode y lo va ejecutando instrucción por instrucción.

🧩 Relación con compiladores

Como tú quieres hacer analizadores (léxico, sintáctico, semántico 👀):

Python funciona así:

```
cpp


Código fuente
   ↓ (parser)
Árbol sintáctico
   ↓
Bytecode
   ↓
Máquina virtual
```

Eso es exactamente un **lenguaje con máquina virtual**, como Java o C#.

Si quieres, puedo mostrar cómo ver el bytecode de un programa Python paso a paso 🔍🐍