# Cours sur l’intérêt et l’utilisation d’un venv

## Table des matières

1. Pourquoi utiliser un environnement virtuel (venv) en Python ?....1
a. Avantages :....1
2. Utiliser un venv sous Debian 13 (ou Linux en général)....2
a. Vérifier que Python et venv sont installés....2
b. Créer un environnement virtuel....2
c. Activer l’environnement virtuel....2
d. Installer des bibliothèques nécessaires au projet dans ce venv....2
e. Vérifier les paquets installés....2
f. Sauvegarder les dépendances....3
g. Recréer un environnement à l’identique grâce à requirements.txt....3
h. Désactiver l’environnement....3
3. Créer une fiche Memento_Python sur l’utilisation des venv....3
4. Retour sur la commande python3 -m venv venv....3
a. python3....3
b. -m....3
c. venv....3
d. venv (le dernier argument)....4
Résultat : que contient le dossier venv ?....4
Pourquoi utiliser -m au lieu de lancer un fichier directement ?....4
En résumé....4

## 1. Pourquoi utiliser un environnement virtuel (venv) en Python ?

Un **environnement virtuel** est un espace isolé qui contient sa propre installation de Python et ses propres bibliothèques.
L’intérêt principal est d’éviter les **conflits entre projets** et de garder ton système propre.

### a. Avantages :

1. **Isolation des dépendances**
   - Chaque projet a ses versions propres de bibliothèques (par exemple `Django 3.2` pour un projet et `Django 5.0` pour un autre).
   - Tu évites les problèmes du type *“mon projet ne marche plus après mise à jour de la bibliothèque numpy”*.
2. **Reproductibilité**
   - Tu peux enregistrer les dépendances dans un fichier `requirements.txt`.
   - Cela permet de recréer le même environnement ailleurs (sur une autre machine, serveur, chez le client ou pour un collègue).
3. **Pas besoin des droits administrateur**

- Tu installes les bibliothèques *localement* dans l'environnement sans impacter Python du système.
- C'est plus sûr et plus propre que d'installer tout en global avec `sudo pip install ...`.

4. **Bonne pratique professionnelle**
   - Dans les équipes, chaque projet est dans son venv.
   - Ça rend ton projet **portable et maintenable**.

## 2. Utiliser un venv sous Debian 13 (ou Linux en général)

### a. Vérifier que Python et venv sont installés

```
python3 --version
```

et installer `venv` si nécessaire (venv est intégré depuis le version 3.3 de python):

```
sudo apt update
sudo apt install python3-venv
```

### b. Créer un environnement virtuel

Dans le dossier `mon_projet` de ton projet, ouvrir un Terminal et tape la commande :

```
user@debian:~/mon_projet$ python3 -m venv venv
```

Ici, le dossier `venv/` créé contient tout l'environnement (interpréteur + librairies).
Tu peux l'appeler autrement, par exemple `.venv` (souvent utilisé pour cacher le dossier).

### c. Activer l'environnement virtuel

```
user@debian:~/mon_projet$ source venv/bin/activate
```

Tu remarqueras que ton invite de commande change ; apparition de (venv). Par exemple :

```
(venv) user@debian:~/mon_projet$
```

Par défaut, c'est le nom du dossier qui contient l'environnement virtuel qui est affiché entre parenthèses. Donc si tu crées et actives un environnement virtuel avec les commandes suivantes :

```
user@debian:~/mon_projet$ python3 -m venv venv2
user@debian:~/mon_projet$ source venv2/bin/activate
(venv2) user@debian:~/mon_projet$
```

Il est possible de différencier le nom de l'environnement virtuel entre parenthèses du nom du dossier qui le contient mais c'est créer une source de confusion. Pour éviter cela, nous ne le ferons jamais et nous ne le présenterons pas ici.

### d. Installer des bibliothèques nécessaires au projet dans ce venv

```
(venv) user@debian:~/mon_projet$ python3 -m pip install pillow
(venv) user@debian:~/mon_projet$ python3 -m pip install flask
```

### e. Vérifier les paquets installés

```
(venv) user@debian:~/mon_projet$ pip list
```

### f. Sauvegarder les dépendances

```
(venv) user@debian:~/mon_projet$ pip freeze > requirements.txt
```

### g. Recréer un environnement à l'identique grâce à requirements.txt

Récupère le fichier `requirements.txt` et copie le dans le dossier où tu veux copier ton projet.

```
 user@debian:~/copie_projet$ python3 -m venv venv
 user@debian:~/copie_projet$ source venv/bin/activate
(venv) user@debian:~/copie_projet$ pip install -r requirements.txt
```

### h. Désactiver l'environnement

Quand tu as fini :

```
(venv) user@debian:~/xxx_projet$ deactivate
 user@debian:~/xxx_projet$
```

## 3. Créer une fiche Memento_Python sur l'utilisation des venv.

## 4. Retour sur la commande `python3 -m venv venv`

La commande suivante est une façon standard de créer un **environnement virtuel Python** :

```
python3 -m venv venv
```

Voyons cela en détail, en particulier le rôle de `-m`.

### a. `python3`

C'est l'interpréteur Python que tu veux utiliser. Sur Debian (et d'autres distributions Linux), `python3` est souvent un alias vers la dernière version installée de Python 3.x.

---

### b. `-m`

L'option `-m` signifie : **"exécuter un module comme un script"**.
En clair, Python va **chercher un module installé** (dans ce cas `venv`) et **l'exécuter comme un programme principal**.
Cela évite d'avoir à connaître le chemin exact du fichier du module. Tu peux faire cela avec n'importe quel module exécutable (comme `http.server`, `pip`, etc.).

---

### c. venv

C'est le nom du **module standard** de Python qui permet de **créer un environnement virtuel** (depuis Python 3.3).
Un **environnement virtuel** est un répertoire isolé qui contient sa propre installation de Python, ainsi que ses propres bibliothèques. C'est très utile pour isoler les dépendances d'un projet.

## d. venv (le dernier argument)

C’est aussi ici le **nom du dossier** dans lequel tu veux créer l’environnement virtuel.
Donc cette commande :

```
python3 -m venv venv
```

va :

- lancer le module `venv` de Python,
- qui va créer un nouvel environnement virtuel dans un dossier appelé `venv` (dans le répertoire courant).

Tu pourrais tout aussi bien l’écrire :

```
python3 -m venv mon_env
```

Et l’environnement serait alors créé dans le dossier `mon_env`.

---

### *Résultat : que contient le dossier venv ?*

Une fois la commande exécutée, tu auras un dossier avec :

- un Python local (dans `venv/bin/python`)
- un gestionnaire de paquets `pip` local (`venv/bin/pip`)
- des répertoires pour les bibliothèques (`venv/lib/...`)
- des scripts d’activation (`venv/bin/activate`)

---

### *Pourquoi utiliser `-m` au lieu de lancer un fichier directement ?*

- **Portabilité** : `-m` fonctionne sur toutes les plateformes sans connaître les chemins spécifiques des fichiers.
- **Fiabilité** : il utilise **le module du Python que tu exécutes**, ce qui évite les confusions si plusieurs versions de Python sont installées.
- **Utilisation de modules standards** : c’est la méthode officielle recommandée pour accéder aux outils Python en ligne de commande.

---

### *En résumé*

| Partie | Signification |
|---|---|
| `python3` | Lance l’interpréteur Python 3 |
| `-m` | Demande à Python d’exécuter un **module** |
| `venv` | Le module à exécuter, ici pour créer un environnement virtuel |
| `venv` (dernier) | Le nom du dossier contenant l’environnement virtuel |