

Installation

L'installation de Python peut se faire selon plusieurs approches, chacune répondant à des besoins spécifiques. Trois méthodes contemporaines se distinguent : **Anaconda**, **Miniconda** et **UV**, cette dernière représentant une avancée notable dans l'écosystème Python moderne.

Anaconda est une distribution complète conçue pour les domaines scientifiques et les applications de data science. Elle fournit un environnement clé en main avec des centaines de bibliothèques préinstallées telles que NumPy, Pandas, Jupyter, Matplotlib ou encore scikit-learn. Cette richesse permet à un utilisateur de démarrer immédiatement un projet complexe sans configuration supplémentaire. Toutefois, ce confort a un coût important en termes d'espace disque (plusieurs gigaoctets) et de lourdeur de gestion. Anaconda impose également l'usage de son propre gestionnaire de paquets **conda**, qui fonctionne en parallèle de **pip** et peut introduire des incohérences ou des lenteurs dans certains contextes.

Miniconda reprend les principes d'Anaconda mais en éliminant toute surcharge initiale. Il fournit uniquement l'essentiel : l'interpréteur Python et le gestionnaire **conda**, laissant à l'utilisateur la responsabilité d'installer manuellement les bibliothèques nécessaires à ses projets. Cette approche offre un bon compromis entre flexibilité et puissance, tout en conservant les avantages de la résolution de dépendances propre à l'écosystème Conda. Toutefois, elle reste tributaire de l'architecture globale d'Anaconda et des limitations inhérentes à **conda**.

UV, enfin, représente une nouvelle génération d'outils dans l'univers Python. Écrit en **Rust**, UV est conçu pour être un gestionnaire de paquets et d'environnements à la fois extrêmement rapide, fiable et minimaliste (Equivalent au **npm** de Node). Il remplace efficacement **pip**, **virtualenv**, et parfois même **venv**, en proposant une installation des dépendances quasi instantanée. Sa conception moderne permet également une meilleure reproductibilité et une intégration facilitée dans des environnements CI/CD ou des conteneurs. De plus, UV n'impose pas de dépendances lourdes et n'engendre pas de surcharge logicielle inutile. Il convient parfaitement aux développeurs recherchant un outil léger, moderne, et performant, tout en restant compatible avec l'écosystème standard de Python (**pyproject.toml**, **requirements.txt**, etc.).

Dans le cadre de cette formation, où la légèreté, la rapidité d'exécution et la maîtrise des dépendances sont essentielles, on adoptera **UV** comme outil d'installation et de gestion d'environnements Python. Ce choix permettra d'aborder les projets avec un environnement épuré, cohérent, et adapté aux exigences contemporaines du développement logiciel.

Installation de UV

Dans le lien suivant, vous trouverez les instructions d'installation de UV pour différents systèmes d'exploitation : [Installer UV](#)

Une seule commande suffit pour l'installer sur Windows avec le terminal PowerShell :

```
powershell -ExecutionPolicy Bypass -c "irm  
https://astral.sh/uv/install.ps1 | iex"
```

Ou sur Mac :

```
curl -LsSf https://astral.sh/uv/install.sh | sh

ou

wget -q0- https://astral.sh/uv/install.sh | sh
```

Pour Windows, une fois l'installation terminée, il vous proposera la commande à lancer pour mettre UV dans vos variables d'environnement :

```
PS C:\WINDOWS\system32> powershell -ExecutionPolicy Bypass -c "irm https://astral.sh/uv/install.ps1 | iex"
Downloading uv 0.8.22 (x86_64-pc-windows-msvc)
Installing to C:\Users\chid-\.local\bin
  uv.exe
  uvx.exe
  uvw.exe
everything's installed!

To add C:\Users\chid-\.local\bin to your PATH, either restart your shell or run:

    set Path=C:\Users\chid-\.local\bin;%Path%    (cmd)
    $env:Path = "C:\Users\chid-\.local\bin;$env:Path"    (powershell)
PS C:\WINDOWS\system32> _
```

Il suffit copier-coller la deuxième commande (Pour powershell) dans la console PowerShell et d'exécuter. (Attention à ne pas sélectionner ce qui est entre parenthèses)

Pour savoir si l'installation s'est bien déroulée, il vous suffit de lancer la commande suivante :

```
PS > uv
```

An extremely fast Python package manager.

Usage: `uv [OPTIONS] <COMMAND>`

Commands:

auth	Manage authentication
run	Run a command or script
init	Create a new project
add	Add dependencies to the project
remove	Remove dependencies from the project
version	Read or update the project's version
sync	Update the project's environment
lock	Update the project's lockfile
export	Export the project's lockfile to an alternate format
tree	Display the project's dependency tree
format	Format Python code in the project
tool	Run and install commands provided by Python packages
python	Manage Python versions and installations
pip	Manage Python packages with a pip-compatible interface
venv	Create a virtual environment
build	Build Python packages into source distributions and wheels
publish	Upload distributions to an index

Initialisation d'un nouvel environnement

Pour commencer à utiliser UV, on va d'abord ouvrir un dossier vide avec VSCode. Ensuite, on va créer un nouvel environnement Python avec la commande suivante :

```
uv init --python 3.12
```

Cette commande va générer les fichiers suivants :

```
bases
├── .gitignore
├── .python-version
├── README.md
├── main.py (ou hello.py)
└── pyproject.toml
```

.gitignore Ce fichier indique à **Git** quels fichiers ou répertoires ne doivent pas être suivis par le système de gestion de version. Dans un projet Python utilisant UV, il est courant d'ignorer les répertoires comme **.venv/**, **__pycache__**/, ou encore les fichiers **.pyc**. Lorsqu'UV crée un environnement virtuel local (par exemple **.venv**), il est recommandé d'exclure ce dossier du suivi Git afin de ne pas versionner les dépendances installées localement.

.python-version Ce fichier contient une simple ligne spécifiant la version de Python que le projet utilise, par exemple `3.12.1`. UV lit ce fichier pour déterminer automatiquement quelle version de l'interpréteur installer ou utiliser si elle est disponible localement. Ce mécanisme est comparable à celui utilisé par des outils comme `pyenv`. Il assure la cohérence de la version Python utilisée à travers différents systèmes et développeurs.

README.md Ce fichier est un document en Markdown servant de point d'entrée documentaire au projet. Il permet de présenter brièvement l'objectif du projet, la manière de l'installer, de l'utiliser, et éventuellement les contributions attendues. Bien qu'il ne soit pas lié à UV en soi, il est une composante essentielle de tout projet structuré, notamment lors du partage sur une forge comme GitHub.

main.py (ou **hello.py**) Il s'agit d'un fichier Python classique contenant du code source. Dans ce contexte, on peut imaginer qu'il contient une fonction simple ou un point d'entrée minimal (comme un `print("Hello, world!")`). Il permet de vérifier que l'environnement Python configuré fonctionne correctement.

pyproject.toml Ce fichier constitue le cœur de la configuration d'un projet Python moderne. Il remplace progressivement le traditionnel `setup.py` et devient la norme pour la gestion des dépendances et de la configuration des outils. Lorsqu'on utilise UV, ce fichier est utilisé pour déclarer :

- le nom du projet,
- les dépendances (`dependencies = [...]`),
- les outils utilisés (comme `tool.uv` ou `tool.pyright`),
- la version minimale de Python requise (`requires-python = ">=3.12"`).

UV lit et gère ce fichier pour installer les bibliothèques, créer l'environnement virtuel, et verrouiller les versions dans un fichier `uv.lock`. C'est aussi ce fichier qui permet une meilleure portabilité du projet entre machines.

Pour lancer le fichier `main.py` avec Python, il suffit d'activer l'environnement virtuel et d'exécuter la commande suivante :

Sur Mac ou Linux :

```
$ uv run main.py
```

Sur Windows :

```
PS > uv run main.py
```

```
C:\Users\chid-\OneDrive\Bureau\bases>uv init --python 3.12
Initialized project `bases`

C:\Users\chid-\OneDrive\Bureau\bases>uv run main.py
Using CPython 3.12.11
Creating virtual environment at: .venv
Hello from bases!
```

Une fois la commande exécutée, vous devriez voir apparaître un dossier **.venv**:

```
bases
├── .gitignore
├── .python-version
├── .venv
│   ├── .gitignore
│   ├── CACHEDIR.TAG
│   ├── bin # (ou Scripts pour Windows)
│   │   ├── activate
│   │   ├── activate.bat
│   │   ├── activate.csh
│   │   ├── activate.fish
│   │   ├── activate.nu
│   │   ├── activate.ps1
│   │   ├── activate_this.py
│   │   ├── deactivate.bat
│   │   ├── pydoc.bat
│   │   ├── python
│   │   ├── python3
│   │   └── python3.12
│   ├── lib
│   │   └── python3.12
│   │       └── site-packages
│   │           ├── __pycache__
│   │           │   └── _virtualenv.cpython-312.pyc
│   │           ├── _virtualenv.pth
│   │           └── _virtualenv.py
│   └── pyvenv.cfg
├── README.md
├── hello.py
├── pyproject.toml
└── uv.lock
```

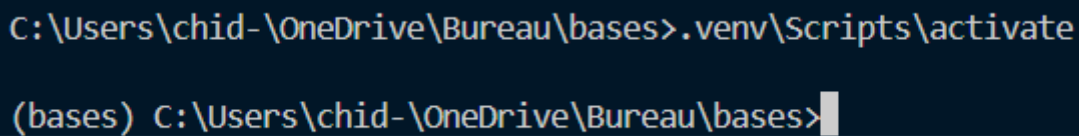
Vous pouvez ainsi activer l'environnement Python avec la commande suivante :

Sur Mac ou Linux :

```
$ source .venv/bin/activate
```

Sur Windows :

```
PS > .venv\Scripts\activate
```



```
C:\Users\chid-\OneDrive\Bureau\bases>.venv\Scripts\activate  
(bases) C:\Users\chid-\OneDrive\Bureau\bases>
```

La présence à gauche de la commande **(bases)** indique que l'environnement est activé.

Le nom de l'environnement virtuel est généré automatiquement par UV. Il prend le nom du dossier parent.

Le dossier **.venv** peut être généré sans avoir à lancer le script Python. En ajoutant par exemple une librairie à notre projet, nous pouvons lancer la commande suivante :

(Supprimer le dossier **.venv** avant)

```
$ uv add numpy  
Using CPython 3.12.7  
Creating virtual environment at: .venv  
Resolved 2 packages in 13.61s  
Prepared 1 package in 679ms  
[0/1] Installing wheels...  
warning: Failed to clone files; falling back to full copy. This may lead  
to degraded performance.  
If the cache and target directories are on different filesystems,  
reflinking may not be supported.  
If this is intentional, set `export UV_LINK_MODE=copy` or use `--  
link-mode=copy` to suppress this warning.  
Installed 1 package in 759ms  
+ numpy==2.3.3
```

Pour résumé voici les commandes à lancer à chaque nouveau projet :

```
uv init --python 3.12  
uv run main.py  
source .venv/bin/activate
```

Pour Windows :

```
PS > uv init --python 3.12  
PS > uv run main.py  
PS > .venv\Scripts\activate
```

Et la commande pour ajouter une librairie :

```
uv add <nom-de-la-librairie>
```

Conclusion

Le choix d'**UV** comme gestionnaire d'environnement et d'installation de dépendances s'inscrit dans une volonté de moderniser et d'optimiser la chaîne de développement Python. Contrairement aux solutions traditionnelles comme Anaconda ou même Miniconda, UV se distingue par sa **légèreté**, sa **rapidité d'exécution**, et sa **philosophie minimaliste**, tout en restant pleinement compatible avec les standards contemporains tels que **pyproject.toml**.

Son installation, extrêmement rapide, permet de disposer en quelques secondes d'un environnement isolé, reproductible, et prêt à l'usage, sans surcharger le système de bibliothèques inutiles. UV centralise la gestion des versions de Python, la résolution des dépendances et la création des environnements, tout en réduisant considérablement le temps d'attente traditionnellement associé à ces opérations.

Dans un contexte pédagogique ou professionnel, l'adoption de UV garantit une montée en compétence fluide, sans friction liée aux outils, et offre un cadre de travail moderne aligné avec les meilleures pratiques actuelles du développement Python. Ce choix stratégique assure une base technique stable, performante et évolutive pour l'ensemble des projets à venir.