



# Frameworks Python

Développement Python et bonnes pratiques

# sommaire

[Environnement virtuel - virtualenv \(venv\)](#)

[Environnement virtuel – venv installation](#)

[Serveur Web - Uvicorn](#)

[Serveur Web – uvicorn - installation](#)

[Serveur Web – gunicorn – production](#)

[Base de données – MySQL](#)

[Base de données - SQLAlchemy](#)

[Questions/Réponses](#)



## Environnement virtuel - virtualenv (venv)

- ***L'utilisation de venv permet de créer un environnement isolé du reste de la configuration système (Python, package, etc. ...)***
- Outil standard intégré à Python depuis la version 3.3
- - Isoler les dépendances de chaque projet
- - Éviter les conflits entre bibliothèques
- - Permettre de tester sans polluer le système
- 💡 Idéal pour un développement propre et reproductible !



# Environnement virtuel – venv installation

- `pip install virtualenv`
- Windows
  - `py -m pip install virtualenv`
- MacOS/Linux
  - `python3 -m pip install virtualenv`


# Environnement virtuel - création

- `python -m venv [venv]` ← (nom de l'environnement)
- Activer l'environnement virtuel :
  - Windows : `.\venv\Scripts\activate.bat` (activate)
  - Mac/Linux : `source venv/bin/activate.sh`
- 🖱️ Désactiver l'environnement virtuel :
  - `...\deactivate.bat` (deactivate)

# Environnement virtuel - utilisation

- Lorsque vous installez un nouveau package, celui-ci n'est installé que sur votre environnement virtuel
- On utilisera ***pip freeze*** pour enregistrer toutes les dépendances du projet en cours :
  - `pip freeze > requirements.txt`
- Pour réinstaller tout l'environnement on pourra utiliser
  - `pip install -r requirements.txt`

## Environnement virtuel - requirements.txt

-  **requirements.txt contient toutes les dépendances d'un projet.**
- - Format : `nom_du_package==version`
- - Facilite la collaboration entre développeurs
- - Permet un déploiement reproductible
- - Exemple :
- `fastapi==0.110.0`
- `uvicorn==0.29.0`





# Serveur Web - **uvicorn**

- **uvicorn** serveur web ASGI léger, ultra rapide, basé sur **asyncio** (non-bloquant, haute performance)
- Il est utilisé pour le développement d'API (Services REST) notamment couplé avec le framework **FastAPI**.
- il reçoit les requêtes des applications clientes, telles que les navigateurs des utilisateurs, et leur envoie les réponses
- - Conçu pour les APIs modernes
- - Supporte HTTP/1.1, WebSocket, HTTP/2 (en option)



# Serveur Web – **uvicorn** - avantages

- Rapide grâce à **uvloop** (event loop optimisée en C)
- - Support natif de **WebSocket**
- - Parfait pour des microservices ou APIs REST
- - Facile à intégrer avec d'autres serveurs comme Gunicorn



# Serveur Web – **uvicorn** - installation

- pip install uvicorn
- OU
- pip install "uvicorn[standard]"
  - Installation complète avec les fonctionnalités avancées
- <https://www.uvicorn.org/>

# Serveur Web – **uvicorn** - lancement

- `uvicorn main:app`

- Lance le serveur web

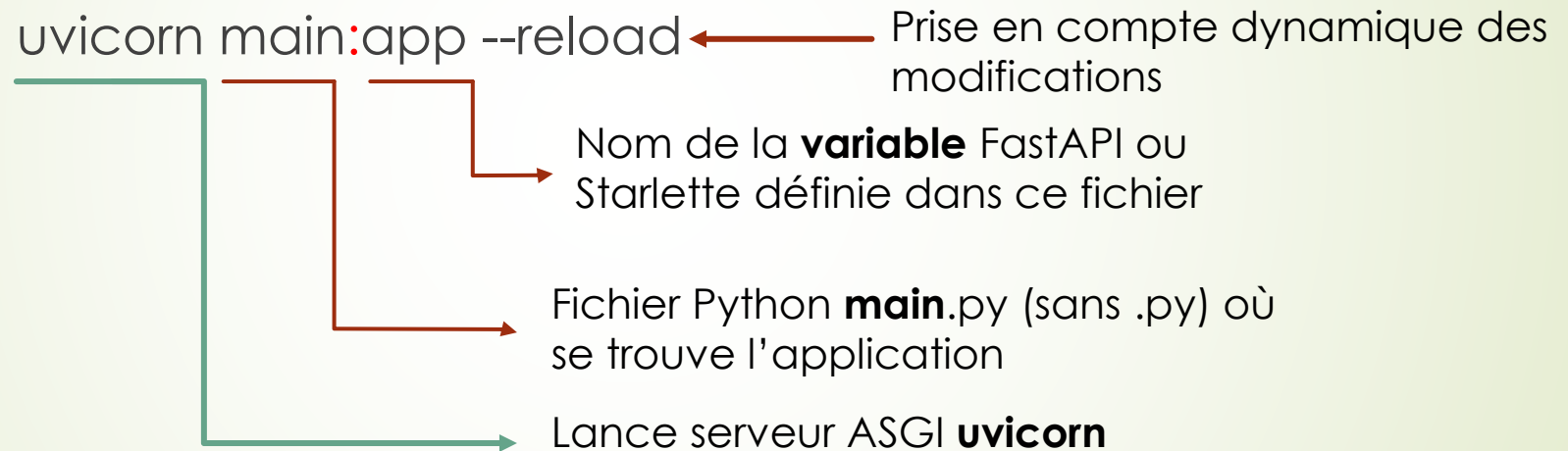
- 🔑 Options utiles :

- `--reload` → rechargement auto (dev)

- `--host 0.0.0.0` → accessible en réseau

- `--port 8000` → port personnalisé

## Serveur Web – **uvicorn** - lancement 2



# Serveur Web – *uvicorn* - exemple

➤ Fichier : `main.py`

```
from fastapi import FastAPI
```

```
app = FastAPI()
```

```
@app.get('/')  
def root():
```

```
    return {'hello': 'world'}
```

➤ ▶ Exécution :

```
uvicorn main:app --reload
```



## Serveur Web – ***gunicorn*** - ***production***

- Pour la mise en production on choisira le serveur Gunicorn
- Il permet le scaling de l'application en travaillant avec des workers pour absorber plus de trafic

# Serveur Web – ***gunicorn*** - *installation*

➤ `pip install "uvicorn[standard]" gunicorn`

Lancement

➤ `gunicorn main:app -w 4 -k uvicorn.workers.UvicornWorker`



Attention ***gunicorn*** ne fonctionne que sur ***Linux***



Contournement sous Windows





# Base de données - MySql

- Pour les applications légères on pourra utiliser l'ORM SQLAlchemy
- Il est très léger et facile à installer
- Il comprend les principales bases de données du marché



# Base de données - SQLAlchemy

- `pip install sqlalchemy pymysql`
  - avec venv activé
- Ensuite on définira un fichier `database.py` qui donnera tous les paramètres de connexion à la base
- On définira un fichier `models.py` où on déclarera les entités (tables) manipulés par l'application

# SQLAlchemy - configuration

➡ models.py

```
from sqlalchemy import Column, Integer, String, DateTime
from sqlalchemy.ext.declarative import declarative_base
Base = declarative_base()
```

***class Country(Base):***

```
    __tablename__ = "country"
    country_id = Column(Integer, primary_key=True, index=True)
    country = Column(String(50), nullable=False)
    last_update = Column(DateTime)
```

# SQLAlchemy - configuration

➡ database.py

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
baseDATABASE_URL =
"mysql+pymysql://username:password@localhost:3306/nom_de_la_base"
engine = create_engine(DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False,
bind=engine)
```

# SQLAlchemy – Application 1

- Dans l'application on déclarera les imports  
***from sqlalchemy.orm import Session***  
***from models import Country***  
***from database import SessionLocal***
- Ouverture d'une session :

```
def get_db():  
    db = SessionLocal()  
    try:  
        yield db  
    finally:  
        db.close()
```

# SQLAlchemy – Application 2

- Accès à la base, via une URL

```
@app.get("/countries")
```

```
def read_countries(db: Session = Depends(get_db)):
```

```
    countries = db.query(Country).all()
```

```
    return [
```

```
        {
```

```
            "country_id": c.country_id,
```

```
            "country": c.country,
```

```
            "last_update": c.last_update
```

```
        }
```

```
        for c in countries
```

```
    ]
```



# Questions/Réponses

Question ?