POETRY GUIDE

IDEO-LAB - 2025

VERSION 1.0

Étape 1 : Installer Poetry

Si vous n'avez pas encore installé Poetry, suivez ces instructions :

1. Ouvrez un terminal (ou une invite de commande).
2. Exécutez la commande suivante pour installer Poetry :

```bash
curl -sSL https://install.python-poetry.org | python3 -
```

3. Ajoutez Poetry à votre chemin (PATH) en suivant les instructions qui s'affichent à la fin de l'installation.

   Vous pouvez aussi vérifier que Poetry est bien installé avec la commande :

```bash
poetry --version
```

Étape 2 : Créer un projet Django avec Poetry

1. Créez un nouveau projet Django avec Poetry :

```bash
poetry new mon_projet_django
```

   Cela créera une structure de projet avec un fichier `pyproject.toml`, qui est utilisé pour gérer les dépendances.

2. Accédez au répertoire de votre projet :

```bash
cd mon_projet_django
```

3. Activez l'environnement virtuel de Poetry :

```bash
poetry shell
```

Étape 3 : Ajouter Django comme dépendance

1. Ajoutez Django à votre projet via Poetry :

   ```bash
   poetry add django
   ```

   Cela installera Django et mettra à jour le fichier `pyproject.toml` pour inclure Django comme dépendance.

Étape 4 : Créer un projet Django

1. Créez un projet Django dans le répertoire :

   ```bash
   django-admin startproject mon_projet .
   ```

2. Vous pouvez maintenant démarrer le serveur de développement Django :

   ```bash
   python manage.py runserver
   ```

   Si tout fonctionne bien, vous verrez un message indiquant que le serveur Django est en cours d'exécution.

Étape 5 : Gérer les dépendances et le fichier `pyproject.toml`

- **Ajouter des dépendances** supplémentaires :

  Par exemple, pour ajouter `djangorestframework`, exécutez :

  ```bash
  poetry add djangorestframework
  ```

- **Installer toutes les dépendances** spécifiées dans le fichier `pyproject.toml` :

  ```bash
  poetry install
  ```

- **Mettre à jour les dépendances** du projet :

  ```bash
  poetry update
  ```

COMMENT UTILISER POETRY

AGENDA

- 1. **Initialisation d’un Projet Poetry**
- 2. **Ajouter des dépendances**
- 3. **Gestion de l’Environnement Virtuel**
- 4. **Installer et Gérer les Dépendances**
- 5. **Vérification des Dépendances**
- 6. **Créer et Publier un Package Python**
- 7. **Gérer des Scripts et Commandes Personnalisées**
- 8. **Gestion des Versions et du Verrouillage des Dépendances**
- 9. **Exporter les Dépendances vers un Fichier**

1. Initialisation d'un Projet Poetry

Poetry simplifie la gestion des dépendances et la création d'environnements virtuels pour vos projets. Pour commencer avec un nouveau projet :

1. **Créer un nouveau projet :**

```bash
poetry new mon_projet
```

Cela crée un répertoire `mon_projet` avec la structure de base :

```markdown
mon_projet/
├── pyproject.toml
├── README.rst
└── mon_projet/
    └── __init__.py
```

Le fichier clé ici est `pyproject.toml`, qui contient toutes les informations de configuration pour votre projet, y compris les dépendances.

2. **Initialiser Poetry dans un projet existant :**

   Si vous avez déjà un projet et que vous voulez utiliser Poetry pour gérer les dépendances, vous pouvez initialiser Poetry dans le projet existant :

   ```bash
   poetry init
   ```

   Vous serez guidé à travers une série de questions pour configurer le fichier `pyproject.toml`.

2. Ajouter des dépendances

L'une des principales fonctionnalités de Poetry est de faciliter l'ajout de bibliothèques à votre projet.

- **Ajouter une bibliothèque :**

  Par exemple, pour ajouter `requests` à votre projet :

  ```bash
  poetry add requests
  ```

  Cette commande fait plusieurs choses :

  - Installe `requests` dans l'environnement virtuel.
  - Met à jour le fichier `pyproject.toml` pour inclure `requests` comme dépendance.
  - Met à jour le fichier `poetry.lock` pour garantir que les mêmes versions de paquets seront utilisées à chaque installation.

- **Ajouter une bibliothèque de développement :**

  Par exemple, pour ajouter `pytest` uniquement pour le développement :

  ```bash
  poetry add --dev pytest
  ```

3. Gestion de l'Environnement Virtuel

Poetry crée automatiquement un environnement virtuel pour chaque projet, ce qui vous permet de garder des environnements isolés pour différents projets.

- **Accéder à l'environnement virtuel :**

  Pour activer l'environnement virtuel et commencer à travailler dans celui-ci :

  ```bash
  poetry shell
  ```

  Cela vous met dans l'environnement isolé où toutes les dépendances spécifiques au projet sont installées.

- **Exécuter une commande dans l'environnement sans entrer dans le shell :**

  Par exemple, pour exécuter un script Python :

  ```bash
  poetry run python script.py
  ```

  Cela exécute `script.py` dans l'environnement virtuel de Poetry sans que vous ayez besoin d'activer manuellement l'environnement.

4. Installer et Gérer les Dépendances

- **Installer toutes les dépendances :**

  Si vous récupérez un projet existant, vous pouvez installer toutes les dépendances définies dans le fichier `pyproject.toml` avec :

  bash Copier le code

  ```
  poetry install
  ```

  Cela installe toutes les bibliothèques, y compris celles de développement si elles sont requises, dans l'environnement virtuel.

- **Mettre à jour les dépendances :**

  Pour mettre à jour toutes les dépendances à leurs dernières versions compatibles (en fonction des contraintes dans `pyproject.toml`), exécutez :

  bash Copier le code

  ```
  poetry update
  ```

  Vous pouvez aussi mettre à jour une seule dépendance :

  bash Copier le code

  ```
  poetry update requests
  ```

5. Vérification des Dépendances

Pour voir toutes les dépendances installées dans votre projet et leurs versions, vous pouvez utiliser la commande :

```bash
poetry show
```

Cela vous donne une liste complète des bibliothèques installées.

6. Créer et Publier un Package Python

Poetry facilite la création de packages que vous pouvez publier sur **PyPI**.

- **Créer un package :**

  Poetry configure déjà votre projet pour être empaqueté dès la création du projet avec `pyproject.toml`. Pour créer un package :

  ```bash
  poetry build
  ```

  Cela génère un fichier **wheel** (`.whl`) et un **source distribution** (`.tar.gz`) dans le répertoire `dist/` .

- **Publier sur PyPI :**

  Si vous avez configuré votre compte PyPI, vous pouvez publier votre package avec la commande :

- **Publier sur PyPI :**

  Si vous avez configuré votre compte PyPI, vous pouvez publier votre package avec la commande :

  bash Copier le code

  ```
  poetry publish
  ```

  Vous pouvez aussi spécifier un dépôt privé si nécessaire.

7. Gérer des Scripts et Commandes Personnalisées

Dans `pyproject.toml`, vous pouvez définir des scripts personnalisés à exécuter via Poetry :

toml Copier le code

```toml
[tool.poetry.scripts]
mon_script = "mon_projet.module:fonction"
```

Cela permet d'exécuter le script directement avec :

bash Copier le code

```bash
poetry run mon_script
```

8. Gestion des Versions et du Verrouillage des Dépendances

Poetry utilise un fichier `poetry.lock` pour verrouiller les versions exactes des bibliothèques installées, ce qui assure la reproductibilité des environnements sur différentes machines.

- **Verrouiller une version spécifique :**

  Si vous voulez spécifier une version particulière pour une dépendance dans `pyproject.toml`, vous pouvez modifier le fichier comme ceci :

  ```toml
  requests = "2.25.1"
  ```

- **Résolution des conflits de dépendances :**

  Si vous rencontrez des conflits entre les bibliothèques, vous pouvez utiliser :

  ```bash
  poetry update --lock
  ```

  Cela tente de résoudre les dépendances et met à jour le fichier `poetry.lock` en conséquence.

9. Exporter les Dépendances vers un Fichier `requirements.txt`

Si vous avez besoin de générer un fichier `requirements.txt` pour un projet, vous pouvez utiliser :

bash Copier le code

```bash
poetry export -f requirements.txt --output requirements.txt
```

Cela génère un fichier `requirements.txt` compatible avec `pip`.

Conclusion

Poetry vous permet de gérer vos dépendances de manière propre et reproductible tout en créant automatiquement des environnements virtuels. Il est également très utile pour la publication de packages Python et offre un flux de travail agréable avec une intégration simple des bibliothèques et des scripts.