Chapter 6 Package

Môi trường ảo (Virtual Environment) — Isolate & Reproducible

Python sử dụng **virtual environment** (ví dụ: venv) để tạo môi trường độc lập, tránh xung đột thư viện giữa dự án. Mỗi env chứa riêng python interpreter, pip và sitepackages riêng (Python Packaging, Python Packaging).

Tạo môi trường:

```
```bash
python3 -m venv .venv
```
```

Môi trường ảo (Virtual Environment) — Isolate & Reproducible

Kích hoạt:

- Unix/macOS: source .venv/bin/activate
- Windows: .venv\Scripts\activate

Sau khi kích hoạt, mọi lệnh pip install, python đều sử dụng môi trường hiện tại (Python Packaging).

Sử dụng deactivate hoặc tắt terminal để thoát env đó.

Quản lý thư viện bằng pip & requirements

pip là package manager tiêu chuẩn của Python, sử dụng kho chính là **PyPI** (Anaconda).

Cài đặt ví dụ:

```
pip install numpy
pip install pandas==1.5.0
pip uninstall somepkg
```

Quản lý dependencies reproducible qua file requirements.txt:

```
pip install -r requirements.txt
```

giúp tái tạo chính xác các thư viện dùng trong project.

Snapshot toàn bộ môi trường

Khi bạn đang ở trong **virtual environment** (đã activate .venv hoặc env tương tự), chạy lệnh:

```
pip freeze > requirements.txt
```

sẽ liệt kê tất cả các **gói đã cài** trong môi trường đó kèm phiên bản, rồi xuất ra file. Đây là cách nhanh và trực tiếp nhất để lấy dependency hiện có (GeeksforGeeks).

Quản lý thư viện và đóng gói dự án

Nên sử dụng requirements.txt với các phiên bản cố định (==) để đảm bảo khả năng tái tạo môi trường giống hệt.

Khi phát triển một thư viện hoặc ứng dụng dạng package, nên có setup.py hoặc pyproject.toml để cài đặt bằng lệnh:

```
pip install -e .
```

Điều này cho phép các thay đổi trong mã nguồn phản ánh trực tiếp vào môi trường đang chạy.

Khi import module, Python sẽ tìm kiếm trong thư mục site-packages của môi trường ảo (venv), phù hợp với mã nguồn bạn đang phát triển.

Cấu trúc thư mục dự án và thiết kế module

Một cấu trúc thư mục hợp lý có thể được như sau:

```
my_project/
    src/
    ___ my_pkg/
             __init__.py
            module1.py
            subpkg/
              — <u>__</u>init__.py
               - submodule.py
    tests/
    └─ test_module1.py
    requirements.txt
    setup.py hoặc pyproject.toml
    README.md
```

Cấu trúc thư mục dự án và thiết kế module

File __init__.py giúp Python nhận diện thư mục đó là một package.

Đặt mã nguồn trong thư mục src/ giúp ngăn việc import nhầm khi chạy từ thư mục gốc (cwd). Khi đó, Python sẽ thêm src/ vào sys.path, đảm bảo import chính xác module.

Thư mục tests/ nên được tổ chức như một package riêng biệt để các công cụ test như pytest dễ dàng phát hiện và xử lý đúng các module.

Cơ chế import và quá trình phân giải module

Khi thực hiện import, Python sẽ gọi __import__() để tìm và nạp module hoặc package tương ứng.

Trình tìm kiếm sử dụng danh sách sys.path để duyệt qua các thư mục theo thứ tự:

- 1. Current working directory (thư mục hiện tại)
- 2. Các thư viện cài trong site-packages của venv
- 3. Các thư viện tiêu chuẩn (standard library)

Cơ chế import và quá trình phân giải module

Ví dụ:

- Khi chạy python main.py từ trong thư mục src/, Python sẽ thêm src/ vào sys.path và có thể import my_pkg.module1 thành công.
- Nếu test được chạy từ một thư mục khác (ví dụ như tests/) mà src/ không có trong sys.path, việc import sẽ bị lỗi.

Một số công cụ như pytest hỗ trợ tự động thêm thư mục chứa module vào sys.path bằng -import-mode=prepend.

Các loại import nên áp dụng

Nên dùng **absolute import** giữa các module trong một package rõ ràng:

```
from my_pkg.subpkg.submodule import MyClass
```

Với module cùng package, nên dùng **relative import rõ ràng** để giúp code dễ tái cấu trúc:

```
from .calc import fn
```

Tránh sử dụng sys.path.append(...) trong mã nguồn chính, trừ trường hợp debug tạm thời. Việc này có thể gây rối loạn trong cơ chế phân giải module của Python.

Cơ chế sâu bên trong: từ import đến đối tượng module

Lần đầu import một module, Python sẽ gọi importlib.__import__() để nạp nội dung module và tạo một đối tượng module trong bộ nhớ.

Kết quả được lưu trong sys.modules để các lần import sau chỉ lấy lại từ cache, không nạp lại từ file.

Cơ chế sâu bên trong: từ import đến đối tượng module

Với các module tiêu chuẩn như math, Python sử dụng loader đặc biệt tên là BuiltinImporter nằm trong sys.meta_path để nạp trực tiếp từ bên trong trình thông dịch.

Nếu bạn xóa toàn bộ sys.meta_path, các module này sẽ không thể import được—điều này cho thấy độ phức tạp và linh hoạt của hệ thống loader trong Python.

Tổng hợp — Luồng hoạt động và khuyến nghị

| Mục tiêu | Cách Python xử lý | Khuyến nghị khi code |
|-------------------------|---|--|
| Tách môi trường | Mỗi venv điều chỉnh sys.path và site-packages riêng | Mỗi dự án dùng một virtual environment |
| Quản lý thư viện | pip + requirements.txt lưu snapshot thư viện | Cố định phiên bản trong requirements.txt |
| Đóng gói và cài
đặt | pip install -e . kết nối code dev với môi trường hiện tại | Cần setup.py hoặc pyproject.toml |
| Import module | Tìm theo sys.path , tạo đối tượng module nếu chưa có | Dùng import tuyệt đối hoặc import tương đối
rõ ràng |
| Test code | Công cụ như pytest có thể inject path riêng | Cấu trúc test thành package riêng +initpy |
| Tránh lộn xộn
import | Thao tác thủ công với sys.path có thể gây lỗi | Hạn chế và chỉ dùng khi debug hoặc testing
đặc biệt |

Kết luận

Môi trường Python nên được cô lập bằng virtualenv để đảm bảo tính nhất quán và tránh xung đột.

Quá trình import trong Python không dựa vào đường dẫn file trực tiếp mà theo cơ chế tìm kiếm trong sys.path, cùng với hệ thống loader như BuiltinImporter, PathFinder, V.V.

Kết luận

Một cấu trúc dự án hợp lý với src/, __init__.py, requirements.txt, và setup.py hoặc pyproject.toml sẽ giúp bạn phát triển, kiểm thử và triển khai ổn định, dễ hiểu, và dễ mở rộng.

Tránh các giải pháp "tạm thời" như chỉnh sửa sys.path trong mã chính. Thay vào đó, hãy dùng các công cụ chuẩn hóa như pip install -e . để Python tự quản lý mọi thứ đúng cách.

cheers

cảm ơn

thank you!

muchas gracias

dziękuję

danke