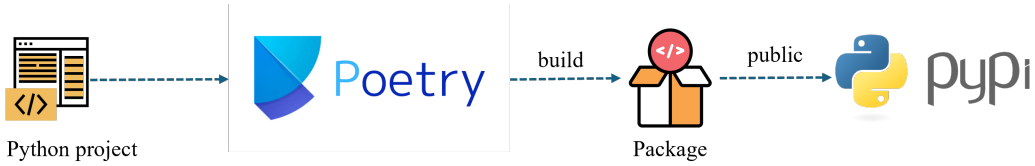
# HƯỚNG DẪN TRIỂN KHAI VÀ ĐÓNG GÓI DỰ ÁN PYTHON VỚI POETRY

*Dinh-Tiem Nguyen và Quang-Vinh Dinh*

## 1 Mở đầu

Làm thế nào để xây dựng các package, library như math, numpy, tensorflow, pytorch? Làm thế nào để tái sử dụng code trong dự án? Trong bài viết này sẽ hướng dẫn cách triển khai xây dựng một package đơn giản với Poetry và Anaconda sau đó phát hành nó trên nền tảng chia sẻ package nổi tiếng PyPI.
Yêu cầu:

- Máy tính đã cài đặt python
- Đã biết cơ bản về cách tạo và quản lí môi trường trong python




Hình 1: Pipline xây dựng package

## 2 Cài đặt thư viện Anaconda và Poetry

Poetry là một công cụ quản lý các package, library phụ thuộc trong môi trường riêng biệt để phát triển các dự án python, đặc biệt là nó cung cấp các chức năng thuận tiện cho việc xây dựng package python và phát hành lên nền tảng PyPI.
Có một số cách khác nhau để cài Poetry, trong hướng dẫn này sử dụng cách cài đặt thông qua pipx - một công cụ để cài đặt và quản lí các package, library python( không phải là các package, library như numpy, pandas mà bạn hay dùng để import khi viết code đâu, mà nó là các package có thể sử dụng từ dòng lệnh như Python, Poetry...) . Để cài đặt pipx, bạn xem hướng dẫn chi tiết tại đây. Đối với hệ điều hành window, các bạn cài đặt qua lệnh sau:

```
py -m pip install --user pipx
```

Tiếp theo chúng ta sẽ cài đặt poetry với lệnh sau:

```
pipx install poetry
```

Sau khi cài đặt hoàn tất, ta có thể xác minh cài đặt thành công không qua lệnh sau:

```
poetry --version
```

Nếu bạn thấy kết quả tương tự như Poetry (version 1.8.2) thì tức là đã cài đặt thành công rồi đó.
Để cài đặt Anaconda, chúng ta có thể xem video hướng dẫn chi tiết tại đây: 40-Day Python Practice: Day 2 - Hướng dẫn tạo môi trường cho Python.

# 3 Thiết lập dự án

Để tạo một dự án mới với Poetry ta sẽ sử dụng lệnh poetry new, ví dụ chúng ta tạo dự án xây dựng package tính giai thừa có tên factorial-aivn, ta sẽ sử dụng lệnh sau:

```
poetry new factorial-aivn
```

Sau khi thực thi lệnh trên, ta sẽ nhận được một thư mục dự án có tên là factorial-aivn với cấu trúc sau:

```
|   pyproject.toml
|   README.md
|
|---factorial_aivn
|       __init__.py
|
|---tests
        __init__.py
```

Trong thư mục factorial-aivn chứa các tệp và package được sử dụng với các mục đích khác nhau:

- factorial_aivn : Ta sẽ viết code để tạo package ở trong thư mục này
- tests : Ta sẽ viết code kiểm thử ở đây
- pyproject.toml : Là file chứa thông tin cấu hình package chúng ta đang xây dựng
- README.md : Là file mô tả package mà chúng ta đang xây dựng, hướng dẫn cài đặt... được viết bằng Markdown

Đó chính là cấu trúc tiêu chuẩn của một dự án xây dựng package với Poetry. Nếu ta có một dự án từ trước mà muốn đóng gói lại với Poetry thì cần phải tổ chức lại code theo cấu trúc này.
Vì file pyproject.toml chứa các thông tin quan trọng nên chúng ta sẽ cùng tìm hiểu nội dung của file này:

```
[tool.poetry]
name = "factorial-aivn"
version = "0.1.0"
description = ""
authors = ["NguyenDinhTiem <nguyendinhtiem1999@gmail.com>"]
readme = "README.md"
packages = [{include = "factorial_aivn"}]

[tool.poetry.dependencies]
python = "^3.12"


[build-system]
requires = ["poetry-core"]
build-backend = "poetry.core.masonry.api"
```

Phần đầu tiên trong file là [tool.poetry], trong phần này chứa thông tin:

- name : Tên của thư mục dự án
- version : Phiên bản mà chúng ta đang phát triển
- description : Mô tả ngắn về package
- authors : Thông tin về tác giả, nhóm phát triển package
- readme : Tệp để viết các mô tả, hướng dẫn chi tiết về package

- packages : Vị trí, tên của package, nơi mà chứa các file code của package.

**Lưu ý**: packages = [{include = "factorial_aivn"}] đây là thông tin vị trí của package, theo mặc định thì poetry sẽ lấy tên theo tool.poetry.name nhưng trong trường hợp dự án của chúng ta có tên thư mục dự án và tên package khác nhau, nên chúng ta cần chỉ định lại tên của package. Nếu không sẽ bị lỗi khi buid package. Vậy nên lần tới phát triển các package khác thì nên đặt tên dự án đơn giản, không nên dùng dấu gạch "-"mặc dù trong tài liệu gốc của Poetry có sử dụng cách này.
Tiếp theo là phần [tool.poetry.dependencies] chứa thông tin về phiên bản python và các package, library tương thích với package chúng ta xây dựng. Ví dụ package chúng ta đang xây dựng cần sử dụng python, streamlit thì chúng sẽ được liệt kê tại đây.
Phần cuối của file là [build-system], phần này chứa một số thông tin về các phụ thuộc để thực hiện đóng gói package. Ta sẽ để các giá trị theo mặc định.
Theo mặc định phiên bản python Poetry sẽ sử dụng để tạo môi trường cho dự án là phiên bản python trên môi trường chung của máy tính. Ta có thể xem thông tin môi trường bằng cách điều hướng đến thư mục dự án vừa tạo và dùng lệnh sau:

```
==========Input==========
cd factorial-aivn
poetry env info
```

```
========== Output ==========
Virtualenv
Python:         3.12.2
Implementation: CPython
Path:           NA
Executable:     NA

Base
Platform:   win32
OS:         nt
Python:     3.12.2
Path:       C:\Python312
Executable: C:\Python312\python.exe
```

Lúc này môi trường chưa được tạo, nhưng theo thiết lập mặc định nó đã chỉ định phiên bản python sử dụng là 3.12. Tuy nhiên thì chúng ta có thể chọn một phiên bản python cụ thể cho dự án bằng cách điều hướng đến vị trí dự án và sử dụng lệnh:

```
poetry env use /full/path/to/python
```

Trong đó /full/path/to/python là đường dẫn đến tệp thực thi python.exe của phiên bản python ta muốn sử dụng. Để có được file này, chúng ta cần cài đặt các phiên bản python khác nhau và có rất nhiều cách để thực hiện điều này. Tuy nhiên, chúng ta sẽ sử dụng Anaconda environment vì nó quá quen thuộc và dễ sử dụng.
Ta tạo môi trường trong Anaconda có tên là factorial_aivn_env với phiên bản python 3.10 với cú pháp dưới đây:

```
conda create -n factorial_aivn_env python=3.10
```

Tiếp theo chúng ta sẽ kích hoạt môi trường factorial_aivn_env vừa tạo bằng lệnh sau:

```
conda activate factorial_aivn_env
```

Để lấy đường dẫn của tệp thực thi python.exe trên window, tại môi trường đang được kích hoạt ta dùng lệnh:

```
==============Input======================
where python
```

```
================ Output ================
C:\Users\Tiem\anaconda3\envs\
    factorial_aivn_env\python.exe
C:\Python312\python.exe
C:\Users\Tiem\.pyenv\pyenv-win\shims\
    python
C:\Users\Tiem\.pyenv\pyenv-win\shims\
    python.bat
C:\Users\Tiem\anaconda3\python.exe
C:\Users\Tiem\AppData\Local\Programs\
    Python\Python311\python.exe
C:\Users\Tiem\AppData\Local\Microsoft\
    WindowsApps\python.exe
========================================
```

Ta sẽ lấy đường dẫn đầu tiên có chứa tên môi trường, sau đó hãy thoát ra khỏi môi trường Anaconda hiện tại bằng cách sử dụng lệnh

```
conda deactivate
```

Tiếp theo ta sử dụng lênh để tạo một môi trường mới với phiên bản python chúng ta đã tạo ra từ môi trường conda. Nhưng trước khi chạy lệnh này, ta cần sửa lại phiên bản python 3.12 trong file pyproject.toml thành 3.10.

```
==============Input======================
poetry env use C:\Users\Tiem\anaconda3\
    envs\factorial_aivn_env\python.exe
```

```
================ Output ================
Creating virtualenv factorial-aivn-
    U849jFk2-py3.10 in C:\Users\Tiem\
    AppData\Local\pypoetry\Cache\
    virtualenvs
Using virtualenv: C:\Users\Tiem\AppData\
    Local\pypoetry\Cache\virtualenvs\
    factorial-aivn-U849jFk2-py3.10
========================================
```

Sau khi thực hiện lệnh trên thì một môi trường có tên factorial-aivn-U849jFk2-py3.10 được tạo ra. Chúng ta có thể kích hoạt môi trường trong Poetry bằng lệnh sau:

```
poetry shell
```

Môi trường này hoạt động tương tự như môi trường trong anaconda. Nhưng điều thú vị là không như Anaconda có thể kích hoạt môi trường ở mọi nơi, môi trường trong Poetry chỉ có thể kích hoạt tại trong chính vị trí thư mục dự án.
Để cài đặt một package hay library cho dự án, ta sẽ sử dụng lệnh sau

```
poetry add name_package
```

Trong ví dụ này chúng ta sẽ cài đặt thư viện streamlit để tạo giao diện, để cài đặt chúng ta sử dụng lệnh:

```
poetry add streamlit
```

Khi chạy lệnh này, poetry sẽ tự động thêm tên package vừa cài đặt vào phần [tool.poetry.dependencies] trong file pyproject.toml. Ta cũng có thể cài nhiều package bằng cách thêm nó vào phần [tool.poetry.dependencies] sau đó sử dụng lệnh sau để cài đặt:

```
poetry install
```

# 4 Xây dựng và public package

## 4.1 Xây dựng package

Chúng ta sẽ xây dựng package tính giai thừa, ngoài ra ta sẽ xây dựng giao diện với streamlit nữa. Trong thư mục factorial_aivn chúng ta sẽ tạo file factorial.py, sau đó viết hàm tính giai thừa như sau.

```
def fact(n):
    """
    Tính giai thừa của một số nguyên dương n.

    Tham số:
    - n: Số nguyên dương.

    Trả về:
    - Giai thừa của n.
    """
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers")
    elif n == 0 or n == 1:
        return 1
    else:
        return n * fact(n-1)
```

Ở đây chúng ta hiểu là mỗi file trong thư mục package được gọi là module nên ta có thể gọi nó trong các module khác. Ta sẽ tạo một file có tên là app.py để viết giao diện tính giai thừa với streamlit.

```
import streamlit as st
from factorial_aivn.factorial import fact

def main():
    st.title("Factorial Calculator")
    number = st.number_input("Enter a number:",
                             min_value=0,
                             max_value= 900)
    if st.button("Calculate"):
        result = fact(number)
        st.write(f"The factorial of {number} is {result}")
        st.balloons()

if __name__ == "__main__":
    main()
```

Và chúng ta hoàn toàn có thể chạy code trong package này bằng cách chạy lệnh sau.

```
streamlit run factorial_aivn/app.py
```

**Lưu ý**: Có thể xuất hiện lỗi tại from factorial_aivn.factorial import fact, khi chạy chương trình tại máy cá nhân thì ta chỉ cần dùng from factorial import fact. Nhưng khi build package thì nhất định phải dùng đường dẫn đầy đủ là from factorial_aivn.factorial import fact

### 4.2 Xây dựng package kiểm thử

Kiểm thử là một phần quan trọng trong việc phát triển dự án, trong phần này chúng ta sẽ thực hiện kiểm thử package qua thư viện Pytest. Trước tiên chúng ta phải cài đặt pytest với lệnh sau:

```
poetry add pytest --group test
```

Trong package tests ta tạo file factorial_test.py với nội dung sau:

```
import pytest
from factorial_aivn.factorial import fact

def test_factorial_of_0():
    assert fact(0) == 1
def test_factorial_of_1():
    assert fact(1) == 1
def test_factorial_of_5():
    assert fact(5) == 120
def test_factorial_of_10():
    assert fact(10) == 3628800
def test_factorial_of_negative_number():
    with pytest.raises(ValueError):
        fact(-5)

if __name__ == "__main__":
    pytest.main()
```

Mỗi hàm trên là một test case, nhằm mục đích kiểm tra xem hàm factorial trong package có hoạt động bình thường không. Chúng ta sử dụng câu lệnh assert để kiểm tra xem kết quả trả về từ hàm factorial có đúng như mong đợi không. Hàm test_factorial_of_negative_number kiểm tra xem hàm factorial có trả về một ngoại lệ ValueError khi được gọi với một số âm không. Cuối cùng, chúng ta gọi pytest.main() để chạy các testcase khi file được chạy như một chương trình độc lập.
Để chạy kiểm thử, chúng ta sử dụng lệnh sau:

```
poetry run pytest -v
```

Chúng ta sẽ nhận được thông tin sau khi thực hiện kiểm thử đã pass 100%, thật tuyệt vời:

```
============================================================================= test session
    starts ==============================================================================
platform win32 -- Python 3.10.14, pytest-8.1.1, pluggy-1.4.0 -- C:\Users\Tiem\AppData\
    Local\pypoetry\Cache\virtualenvs\factorial-aivn-U849jFk2-py3.10\Scripts\python.exe
cachedir: .pytest_cache
rootdir: D:\AIVIETNAM\Asisstant\Thang3\Day28\factorial-aivn
configfile: pyproject.toml
collected 5 items

tests/factorial_test.py::test_factorial_of_0 PASSED
                                                                                [ 20%]
tests/factorial_test.py::test_factorial_of_1 PASSED
                                                                                [ 40%]
tests/factorial_test.py::test_factorial_of_5 PASSED
                                                                                [ 60%]
tests/factorial_test.py::test_factorial_of_10 PASSED
                                                                                [ 80%]
tests/factorial_test.py::test_factorial_of_negative_number PASSED
                                                                                [100%]

============================================================================== 5 passed in
    0.02s ===============================================================================
```

**Lưu ý**: Với test case tính giai thừa cho số lớn 1000 sẽ bị lỗi, do đây là hướng dẫn đơn giản nhằm mục đích chính là xây dựng package nên không quá khắt khe về việc bắt lỗi. Bạn đọc có thể cải tiến thuật toán và thêm nhiều test case khác.

## 4.3 Đóng gói và Xuất bản package

### 4.3.1 Đóng gói

Để đóng gói package để cung cấp cho các nhóm phát triển khác trong dự án hoặc để đăng tải trực tuyến cho cộng đồng python có thể sử dụng. Trong Poetry chúng ta có thể thực hiện điều này rất dễ dàng. Đầu tiên chúng ta đóng gói lại dự án bằng lệnh sau:

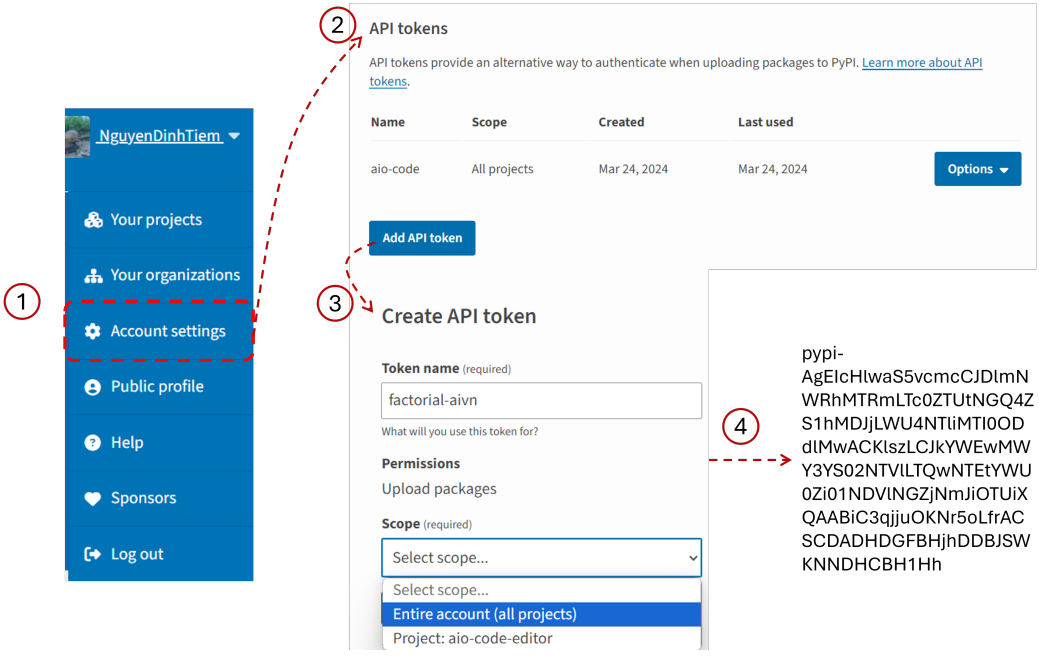
```
==========Input==========
poetry build
```

```
================= Output =================
Building factorial-aivn (0.1.0)
  - Building sdist
  - Built factorial_aivn-0.1.0.tar.gz
  - Building wheel
  - Built factorial_aivn-0.1.0-py3-none-
    any.whl
==========================================
```

Việc đóng gói này sẽ sử dụng các cấu hình mà chúng ta thiết lập trong file pyproject.toml. Vậy là quá trình đóng gói package của chúng ta đã thành công.

### 4.3.2 Xuất bản package

Tiếp theo, để đăng tải lên PyPI, ta sẽ truy cập vào trang web pypi.org và tiến hành đăng nhập, nếu chưa có tài khoản chúng ta sẽ cần phải đăng ký một tài khoản mới.




Hình 2: Tạo PyPI API token

Sau khi đăng nhập thành công, ta sẽ tạo một API token tại phần Account settings (1) hoặc tại link https://pypi.org/manage/account/token/. Sau đó tại phần API tokens chọn Add API tokens(2)phần

Scope chúng ta chọn Entire account (all projects) như hình trên(3).
**Lưu ý**: Trước khi tạo token PyPI sẽ yêu cầu chúng ta xác thực Two factor authentication (2FA). Chúng ta cần làm theo thông báo đó để bật xác thực 2FA, sau đó mới có thể tạo token được.
(4)Cuối cùng chúng ta sẽ nhấn vào button create token, một đoạn mã chứa thông tin về token xuất hiện, ta cần copy lại và lưu vào một tệp nào đó trên máy tính của chúng ta để có thể xem lại khi cần. Vì đoạn mã này chỉ xuất hiện một lần duy nhất khi chúng ta tạo token trên PyPI.
Tiếp theo ta sẽ thêm tài khoản PyPI vào Poetry với lệnh sau:

```
poetry config http-basic.foo <username> <password>
```

Trong đó username là tên đăng nhập, password là mật khẩu của tài khoản PyPI mà chúng ta đã tạo.
Tiếp theo chúng ta thêm thông tin token của tài khoản bằng lệnh:

```
poetry config pypi-token.pypi <my-token>
```

Trong đó my-token là token ta đã tạo ở phía trên.
Cuối cùng chúng ta đẩy package của chúng ta lên nền tảng PyPI bằng lệnh:

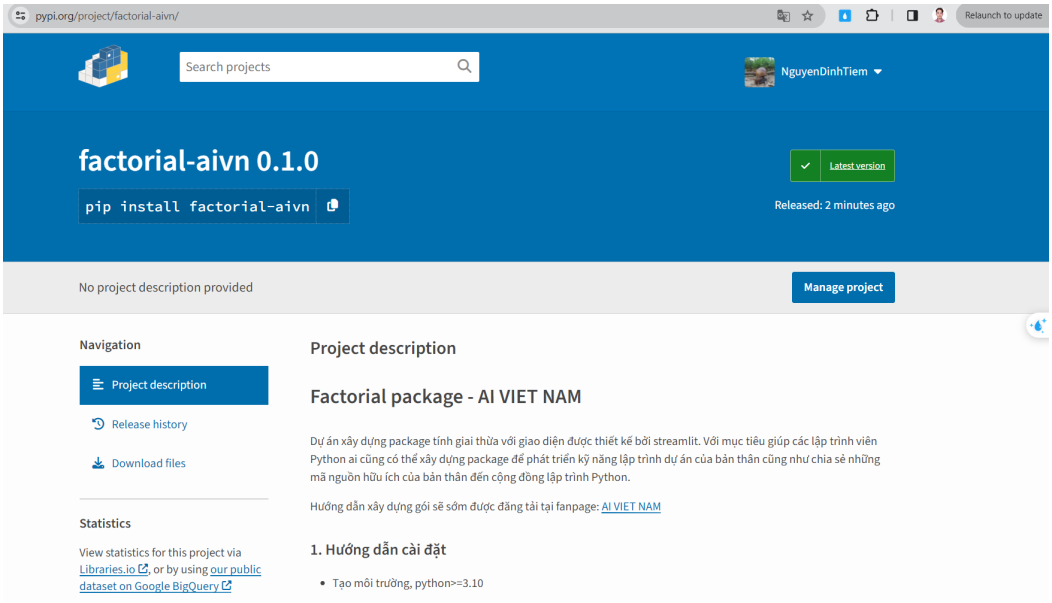
```
==========Input==========
poetry publish




=========================
```

```
================= Output =================
Publishing factorial-aivn (0.1.0) to PyPI
 - Uploading factorial_aivn-0.1.0-py3-none
   -any.whl 100%
 - Uploading factorial_aivn-0.1.0.tar.gz
   100%
==========================================
```

Vậy là package của chúng ta đã được đăng tải lên PyPI, ngay lúc này cả cộng đồng Python có thể sử dụng packge của chúng ta thông qua cú pháp pip install, thật là tuyệt vời ông mặt trời phải không?




Hình 3: Package được đăng tải lên PyPI