



Lập trình Python cơ bản

Bài 9: Module – Package



Phòng Lập Trình - Mạng

Nội dung

- 1. Module**
- 2. Namespace**
- 3. Package**
- 4. Một số module sẵn có trong Python**
- 5. Cài đặt và quản lý Package với PIP**

1. Module

- ❑ **Module** là một tập tin chứa code Python. Trong module ta có thể định nghĩa function, class, variable, thậm chí thể chèn (include) runnable code.
- ❑ Việc nhóm các code có liên quan vào chung 1 module giúp cho code dễ hiểu và dễ sử dụng.
- ❑ Người lập trình có thể tham chiếu tới các module đã được xây dựng trước đó.

1. Module

□ Phân loại:

- Viết bằng Python: có phần mở rộng là **.py**
- Các thư viện liên kết động: có phần mở rộng là **.dll, .pyd, .so, .sl, ...**
- C-Module liên kết với trình phiên dịch.



1. Module

□ Khai báo và sử dụng module:

- Giả sử đã có file **my_module.py** với nội dung như sau:

```
def cong(a, b):
    return a + b

def tru(a, b):
    return a - b

def nhan(a, b):
    return a * b

def chia(a, b):
    if (b != 0):
        return a/b
    else:
        return 0
```

1. Module

□ Khai báo và sử dụng module:

- Sau đó, tạo một file có tên **my_program.py**, trong cùng thư mục với file **my_module.py** ở trên, có nội dung như sau. Lưu ý tên file trong lệnh **import** không có phần mở rộng (**my_module**):

```
import my_module  
x, y = 10, 5  
print('%i + %i = %i' % (x, y, my_module.cong(x, y)))
```

10 + 5 = 15

1. Module

□ Cú pháp:

- Dạng 1: **import module_name**
 - Sẽ import tất cả các **functions, properties** có trong **module_name** vào script.
 - Khi sử dụng **functions/properties** phải kèm tên module đi liền trước, ví dụ **print(math.abs(-3))**.

1. Module

❑ Cú pháp:

- Dạng 2:

```
from module_name import functions_name/properties_name
```

- Chỉ import functions, properties cần dùng có trong module_name vào script. Do đó nếu muốn sử dụng dạng 2 để import tất cả các functions, properties có trong module_name vào script, ta sử dụng cú pháp sau: **from module_name import ***
- Khi sử dụng functions/properties không cần kèm tên module đi liền trước, ví dụ *print(abs(-3))*.

1. Module

□ Lưu ý:

- Lệnh *import* hoặc *from ...* phải được đặt ở đầu script. Lưu ý tên file trong lệnh *import* không có phần mở rộng (**my_module**).
- *Module* chỉ được *load* 1 lần và không phụ thuộc vào số lần được *import*.



1. Module

□ Thú tự tìm kiếm module của trình thông dịch:

Khi chương trình có *import* một *module* nào đó, trình thông dịch sẽ tiến hành tìm kiếm file module tương ứng theo thứ tự thư mục sau:

1. Thư mục hiện hành mà script đang gọi.
2. Các thư mục trong **PYTHONPATH** (nếu có set)
3. Các thư mục cài đặt mặc định (`/usr/local/lib/python` trên Linux/Unix).
4. Module tìm kiếm đường dẫn được lưu trữ trong *module* hệ thống `sys` có tên `sys.path variable`.
Trong `sys.path variable` chứa thư mục hiện hành, **PYTHONPATH** và những cài đặt phụ thuộc mặc định (*installation-dependent default*)
5. Nếu không tìm thấy *module* thì báo lỗi **ImportError**.

1. Module

□ Biến `__name__`:

- Trong Python, một chương trình hoặc một *module* nhỏ trong một chương trình lớn thì mã nguồn sẽ được lưu dưới dạng là một file có đuôi mở rộng là `.py`.
- Khi cần sử dụng file mã nguồn này, ta có 2 cách:
 - Cách 1: Thực thi mã nguồn Python trực tiếp bằng câu lệnh *console* của hệ điều hành (*Command Line*) hoặc trong màn hình của ứng dụng *VSCode*, right click vào chương trình đang có rồi chọn *Run File in Python Terminal*.
 - Cách 2: *import* mã nguồn Python vào trong một file mã nguồn Python khác.

1. Module

□ Biến `__name__`:

- Biến `__name__`: khi đoạn mã thực thi, có thể người lập trình cần kiểm soát đoạn mã lệnh đang thực hiện được chứa trong file Python nào? Vì vậy, Python cung cấp biến `__name__` với 2 dạng giá trị như sau:
 - Dạng 1: nếu file Python được thực thi trực tiếp bằng *Command Line* thì biến `__name__` sẽ có giá trị là "`__main__`".
 - Dạng 2: nếu file Python được *import* thành *module* của chương trình Python khác thì giá trị của biến `__name__` sẽ là **tên của file Python** đang chứa module đó.

1. Module

□ Biến __name__:

- Ví dụ:

- my_module.py

```
def tinh_binh_phuong(so):
    print('Đang gọi thư viện my_module, biến __name__ =', __name__)
    return so ** 2
```

- my_program.py

```
import my_module
print('Đang run my_program.py, biến __name__ =', __name__)
print(my_module.tinh_binh_phuong(8))
```

1. Module

□ Biến `__name__`:

- Ví dụ:

- Kết quả:

Đang run `my_program.py`, biến `__name__ = __main__`

Đang gọi thư viện `my_module`, biến `__name__ = my_module`

64



1. Module

❑ Câu lệnh `if __name__ == '__main__':`

- Ví dụ 1: Thường được dùng khi người lập trình muốn một số đoạn code chỉ được thực thi khi bạn chạy trực tiếp bằng *Command Line* (cách 1) mà không được thực thi khi được *import* thành *module* (cách 2).

1. Module

□ Câu lệnh `if __name__ == '__main__':`

- Ví dụ 1:

- `my_module.py`

```
def tinh_binh_phuong(so):
    print('Đang gọi thư viện my_module, biến __name__ =', __name__)
    if __name__ == '__main__':
        return so ** 2
    else:
        return so
```

- `my_program.py`

```
import my_module
print('Đang run my_program.py, biến __name__ =', __name__)
print(my_module.tinh_binh_phuong(8))
```

1. Module

□ Câu lệnh `if __name__ == '__main__':`

- Ví dụ 1:

- Kết quả:

Đang run my_program.py, biến `__name__ = __main__`

Đang gọi thư viện my_module, biến `__name__ = my_module`

8



1. Module

❑ Câu lệnh `if __name__ == '__main__':`

- Ví dụ 2: Lấy lại ví dụ trên nhưng cả 2 file `my_module.py` và `my_program.py` đều có hàm `tinh_binh_phuong()` (2 hàm trùng tên và ở 2 file .py khác nhau). Theo thứ tự ưu tiên thì hàm `tinh_binh_phuong()` trong file `my_program.py` được thực hiện.

1. Module

□ Câu lệnh `if __name__ == '__main__':`

- Ví dụ 2:

- `my_module.py`

```
def tinh_binh_phuong(so):
    print('Đang gọi thư viện my_module, biến __name__ =', __name__)
    if __name__ == '__main__':
        return so ** 2
    else:
        return so
```

1. Module

□ Câu lệnh `if __name__ == '__main__':`

- Ví dụ 2:

- `my_program.py`

```
from my_module import tinh_binh_phuong
def tinh_binh_phuong(so):
    if __name__ == '__main__':
        return so ** 3
    else:
        return so

print('Đang run my_program.py, biến __name__ =', __name__)
print(tinh_binh_phuong(8))
```

1. Module

□ Câu lệnh `if __name__ == '__main__':`

- Ví dụ 2:

- Kết quả:

Đang run my_program.py, biến __name__ = __main__
512



1. Module

□ Xem thông tin về module:

- Xem thông tin về các hàm, phương thức, ... có trong module:

- Cú pháp: `dir(module_name)`

```
import calendar  
print(dir(calendar))
```

```
import datetime  
print(dir(datetime))
```

- Xem hàm/phương thức thuộc module nào

- Sử dụng phương thức `getmodule` trong module `inspect`

```
from inspect import getmodule  
from math import sqrt  
from my_module import cong  
print(getmodule(sqrt)) # <module 'math' (built-in)>  
print(getmodule(cong)) # <module 'my_module' from '<path>\\my_module.py'>
```

Nội dung

1. Module
2. Namespace
3. Package
4. Một số module sẵn có trong Python
5. Cài đặt và quản lý Package với PIP

2. Namespace

- ❑ **Variable** là tên/định danh (*identifier*) ánh xạ đến object.
- ❑ **Namespace** là một thư mục của các *variable name* (*key*) và các đối tượng tương ứng (*value*).
- ❑ *Statement* có thể truy xuất variable trong *local namespace* và trong *global namespace*.
Nếu *local* và *global variable* có cùng tên với nhau, sẽ ưu tiên cho *local variable*.
- ❑ Mỗi *function* có *local namespace* riêng. Phương thức của *class* có các quy tắc xác định phạm vi tương tự như *function*.

2. Namespace

- Để xem các tên được định nghĩa trong *module*, ta dùng *dir()* hoặc thuộc tính *__dict__*.

```
import math
print(dir(math))

['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2',
'atanh', 'ceil', 'comb', 'copysign', 'cos', 'cosh', 'degrees', 'dist', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs',
'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf',
'isnan', 'isqrt', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'perm', 'pi', 'pow', 'prod',
'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

2. Namespace

- Để xem các tên được định nghĩa trong *module*, ta dùng *dir()* hoặc thuộc tính *__dict__*.

```
import math  
print(math.__dict__)
```

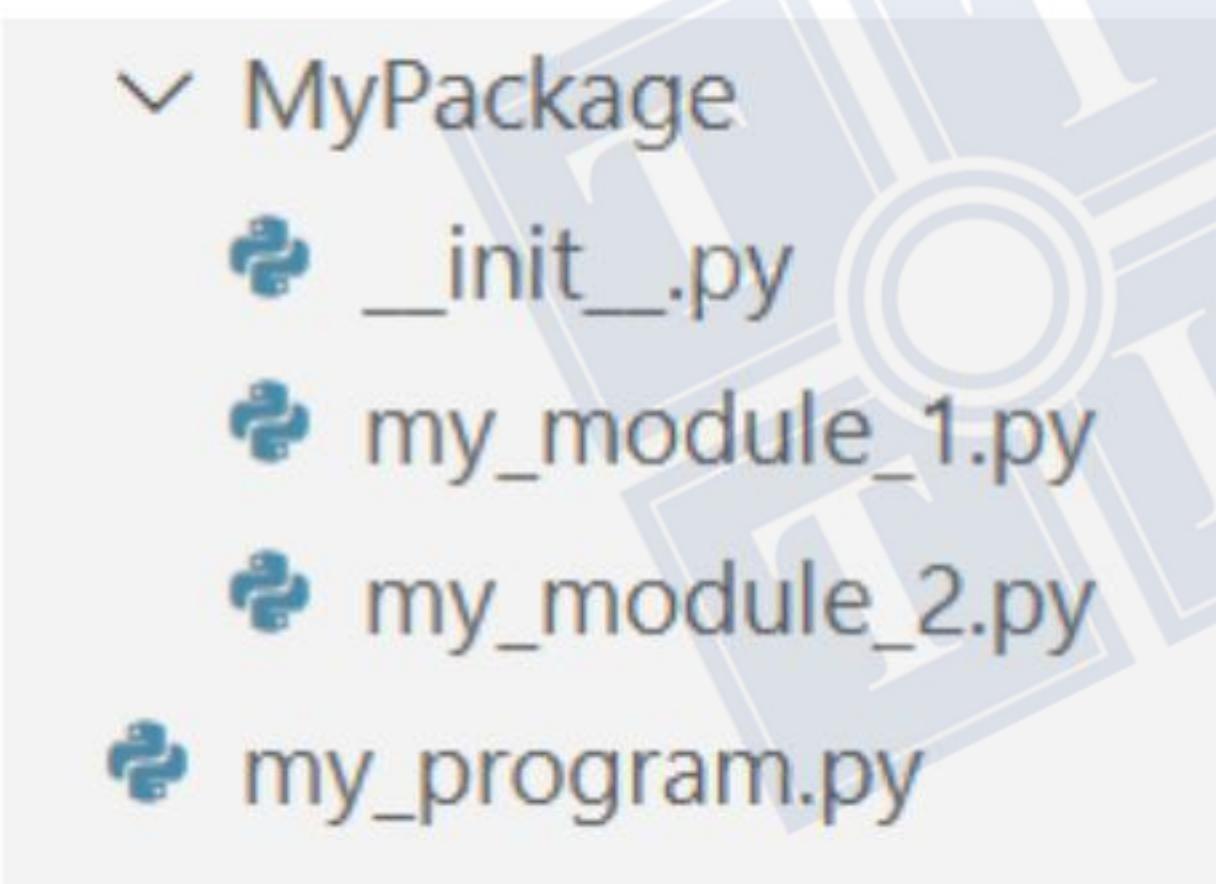
```
{'__name__': 'math', '__doc__': 'This module provides access to the mathematical functions\ndefined by the C standard.', '__package__': '', '__loader__': <class '_frozen_importlib.BuiltinImporter'>, '__spec__': ModuleSpec(name='math', loader=<class '_frozen_importlib.BuiltinImporter'>, origin='built-in'), 'acos': <built-in function acos>, 'acosh': <built-in function acosh>, 'asin': <built-in function asin>, 'asinh': <built-in function asinh>, 'atan': <built-in function atan>, 'atan2': <built-in function atan2>, 'atanh': <built-in function atanh>, 'ceil': <built-in function ceil>, 'copysign': <built-in function copysign>, 'cos': <built-in function cos>, 'cosh': <built-in function cosh>, 'degrees': <built-in function degrees>, 'dist': <built-in function dist>, 'erf': <built-in function erf>, 'erfc': <built-in function erfc>, 'exp': <built-in function exp>, 'expm1': <built-in function expm1>, 'fabs': <built-in function fabs>, 'factorial': <built-in function factorial>, 'floor': <built-in function floor>, 'fmod': <built-in function fmod>, 'frexp': <built-in function frexp>, 'fsum': <built-in function fsum>, 'gamma': <built-in function gamma>, 'gcd': <built-in function gcd>, 'hypot': <built-in function hypot>, 'isclose': <built-in function isclose>, 'isfinite': <built-in function isfinite>, 'isinf': <built-in function isinf>, 'isnan': <built-in function isnan>, 'isqrt': <built-in function isqrt>, 'ldexp': <built-in function ldexp>, 'lgamma': <built-in function lgamma>, 'log': <built-in function log>, 'log1p': <built-in function log1p>, 'log10': <built-in function log10>, 'log2': <built-in function log2>, 'modf': <built-in function modf>, 'pow': <built-in function pow>, 'radians': <built-in function radians>, 'remainder': <built-in function remainder>, 'sin': <built-in function sin>, 'sinh': <built-in function sinh>, 'sqrt': <built-in function sqrt>, 'tan': <built-in function tan>, 'tanh': <built-in function tanh>, 'trunc': <built-in function trunc>, 'prod': <built-in function prod>, 'perm': <built-in function perm>, 'comb': <built-in function comb>, 'pi': 3.141592653589793, 'e': 2.718281828459045, 'tau': 6.283185307179586, 'inf': inf, 'nan': nan}
```

Nội dung

1. Module
2. Namespace
3. Package
4. Một số module sẵn có trong Python
5. Cài đặt và quản lý Package với PIP

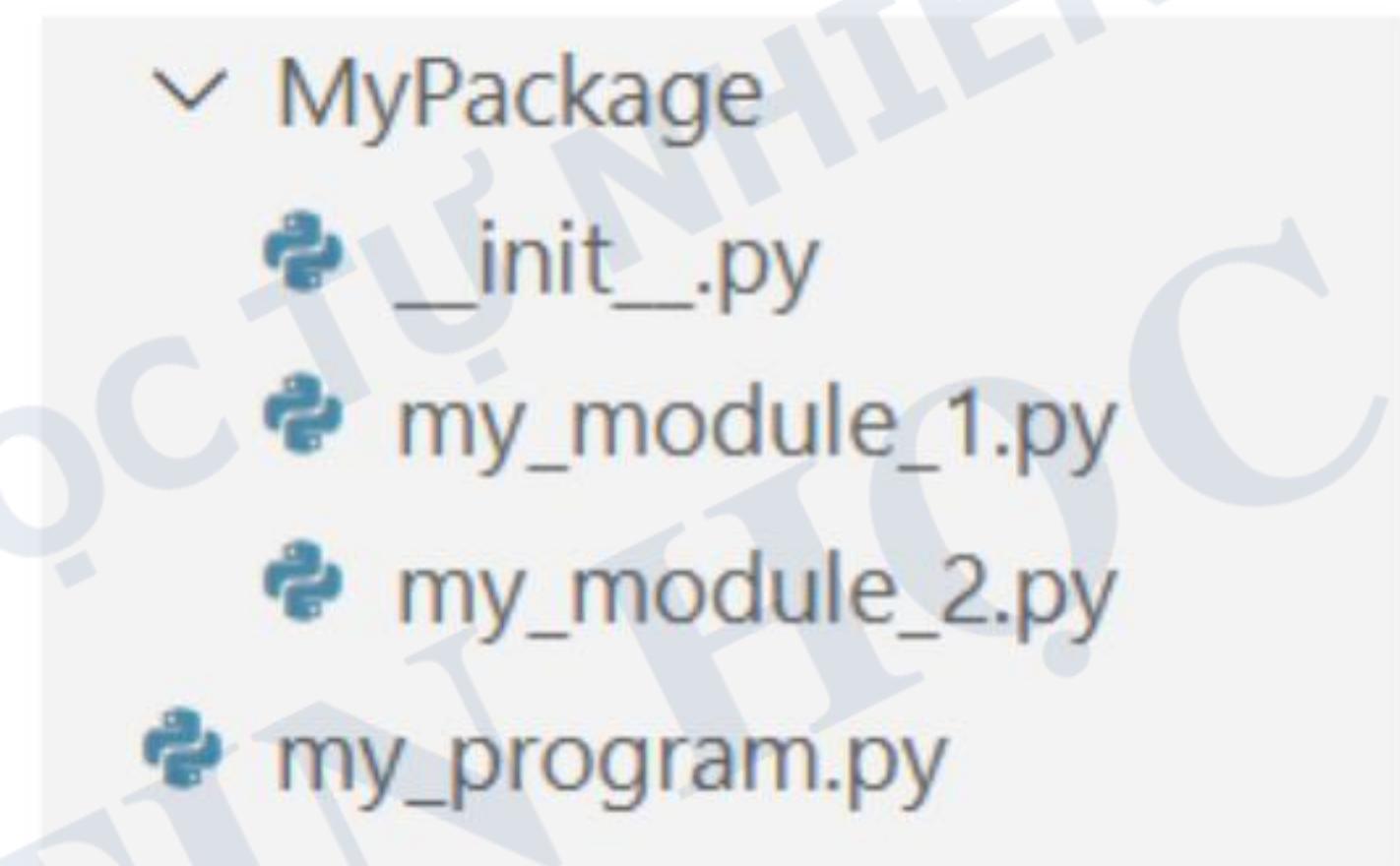
3. Package

- ❑ **Package** là cấu trúc dạng thư mục, nơi chứa các module, subpackage,... trong ứng dụng Python.
- ❑ **Package module:**
 - Có thể gom nhiều *module* (*tập tin .py*) vào một thư mục và **tên thư mục** là tên của **package**. Sau đó tạo một file **`__init__.py`** trong thư mục này.
 - Như vậy, cấu trúc thư của một **package** sẽ như sau:



3. Package

- ❑ Từ *my_program.py* có thể gọi module bằng cách:
 - `import MyPackage.my_module_1`
 - Hoặc `import MyPackage.my_module_1 as md1`
- ❑ Khi sử dụng một module thuộc một package thì các lệnh trong file `__init__.py` sẽ được thực hiện trước. Thông thường thì file `__init__.py` sẽ rỗng.
- ❑ Có thể tạo các subpackage bên trong một package theo đúng cấu trúc thư mục, có file `__init__.py`.



Nội dung

1. Module
2. Namespace
3. Package
4. Một số module sẵn có trong Python
5. Cài đặt và quản lý Package với PIP

4. Một số module sẵn có trong Python

- Thư viện chuẩn của Python rất lớn, cung cấp rất nhiều các phương thức, chức năng... phục vụ cho việc viết code.
- Tham khảo: <https://docs.python.org/3/library/>



Nội dung

1. Module
2. Namespace
3. Package
4. Một số module sẵn có trong Python
5. Cài đặt và quản lý Package với PIP

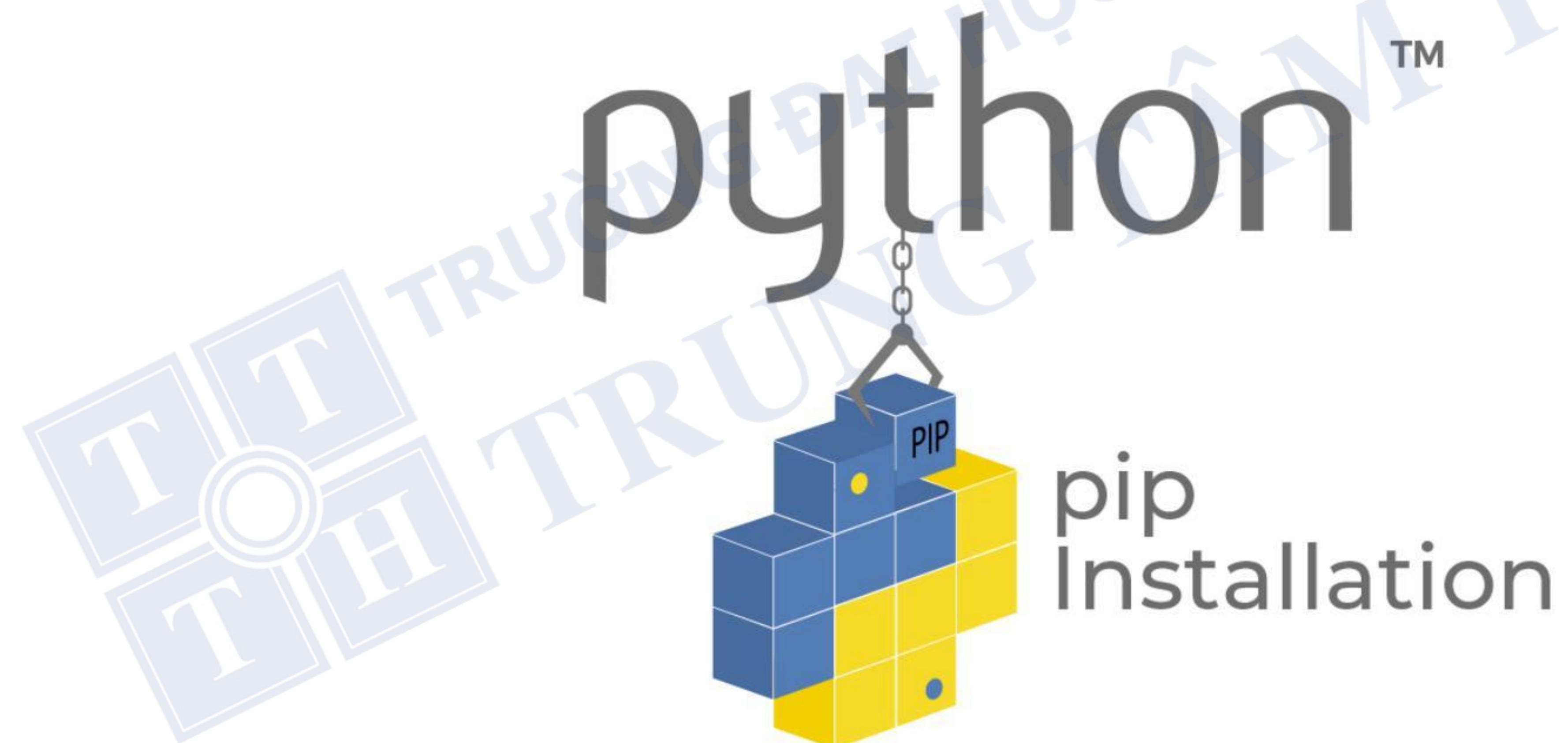
5. Cài đặt và quản lý Package với PIP

- ❑ Bên cạnh việc Python đi kèm với một số module cơ bản được tích hợp sẵn như *math*, *random*, *time*, *calendar*, *date*, *datetime*,... Python còn hỗ trợ các thư viện và framework của bên thứ ba cực kỳ hữu ích giúp chúng ta tránh việc mất công viết lại từ đầu.
- ❑ Các thư viện được gọi là **PyPI (Python Packages Index)**: <https://pypi.org/>

5. Cài đặt và quản lý Package với PIP

❑ PIP là gì?

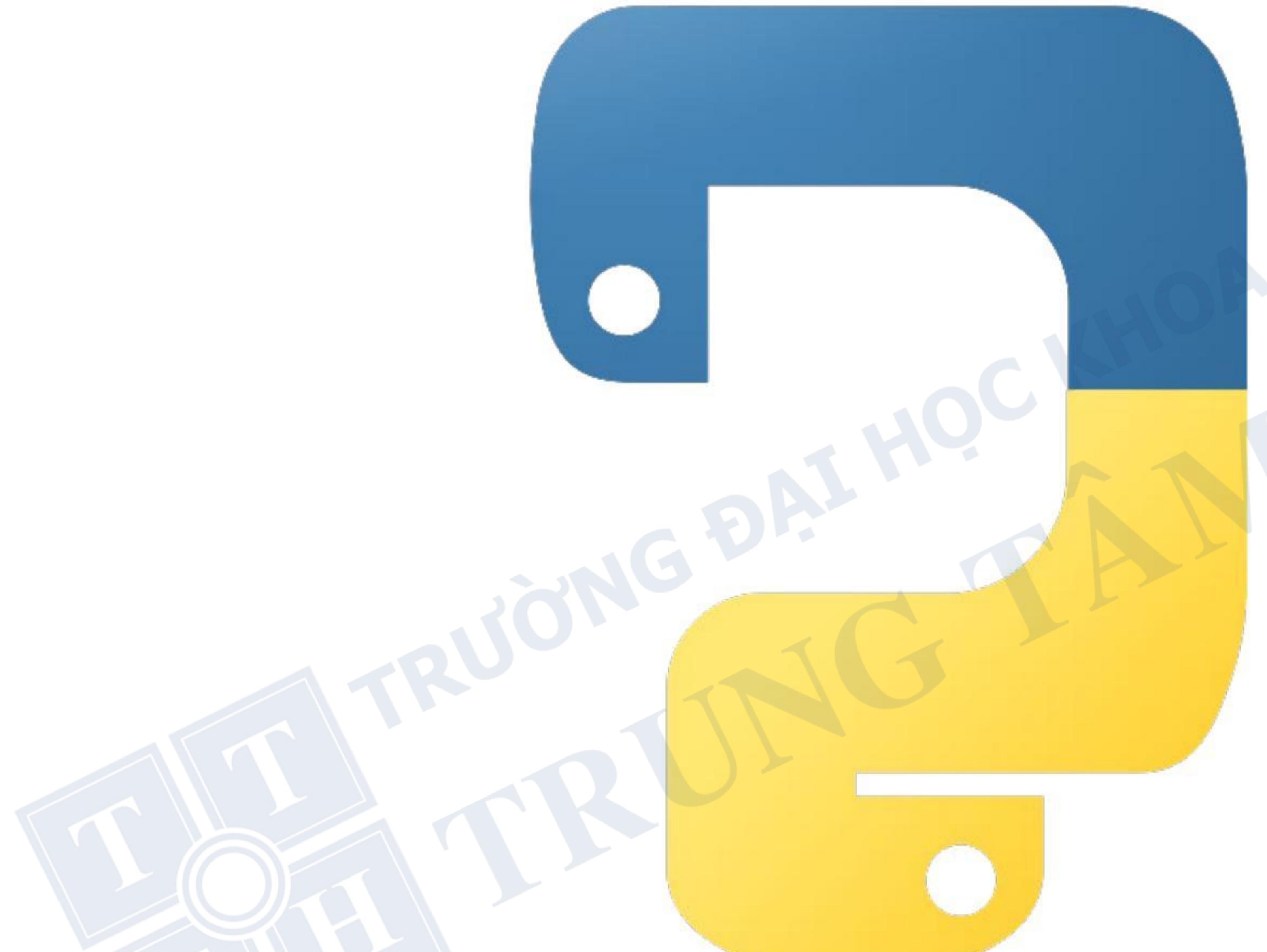
- *PIP là một trình quản lý thư viện cho Python, viết tắt của từ Preferred Installer Program. Đây là một tiện ích dòng lệnh cho phép bạn cài đặt, cài đặt lại hoặc gỡ cài đặt các gói PyPI.*



5. Cài đặt và quản lý Package với PIP

❑ Một số lệnh PIP cơ bản:

- Cài đặt package: **pip install <package_name>**
- Xem chi tiết package đã cài đặt: **pip show <package_name>**
- Liệt kê tất cả các package đã cài đặt: **pip list**
- Liệt kê tất cả các package đã lỗi thời: **pip list --outdated**
- Nâng cấp package đã lỗi thời: **pip install <package_name> --upgrade**
- Gỡ bỏ package: **pip uninstall <package_name>**
- ...



TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
TIT TRUNG TÂM TIN HỌC
TTH