# TÁC GIẢ: 8 SYNC

## CỘNG ĐỒNG

- Kevin Nguyễn
- Nhóm Chia Sẻ Công Nghệ
- Nhóm BlockChain
- Tiktok: 8 Sync
- Youtube: 8 Sync Dev
- Zalo

## KHÓA HỌC:

- Fullstack Python
- Fullstack Nextjs
- Fullstack Android-IOS

Tài liệu sẽ được cập nhật định kì và thông báo trong group nên các bạn chú ý nhen .

# PYTHON - MODULE (MODULE PYTHON)

## GIỚI THIỆU VỀ MODULE PYTHON

Trong Python, module là một tập tin chứa các định nghĩa của các hàm, lớp, biến, hằng số hoặc bất kỳ đối tượng Python nào khác. Nội dung của tập tin này có thể

được sử dụng trong bất kỳ chương trình nào khác. Python cung cấp từ khóa `import` để thực hiện điều này.

## VÍ DỤ VỀ MODULE PYTHON

< 8 Sync Dev />

```
import math
print("Căn bậc hai của 100:", math.sqrt(100))
```

Kết quả:

< 8 Sync Dev />

```
Căn bậc hai của 100: 10.0
```

## CÁC MODULE SẴN CÓ TRONG PYTHON

Thư viện chuẩn của Python đi kèm với một số lượng lớn các module, được gọi là các module sẵn có. Các module này cung cấp các chức năng hữu ích như quản lý hệ điều hành cụ thể, đọc ghi tệp, mạng lưới, v.v.

Dưới đây là một số module sẵn có quan trọng:

1. **os**: Cung cấp một giao diện thống nhất cho một số chức năng hệ điều hành.
2. **string**: Chứa một số hàm để xử lý chuỗi.
3. **re**: Cung cấp một tập hợp các tính năng biểu thức chính quy mạnh mẽ.
4. **math**: Thực hiện các phép toán số học cho các số dấu chấm động.
5. **cmath**: Chứa các phép toán số học cho các số phức.
6. **datetime**: Cung cấp các hàm để làm việc với ngày tháng và thời gian trong một ngày.
7. **gc**: Cung cấp một giao diện cho bộ thu gom rác tích hợp.

8. **asyncio**: Xác định các chức năng cần thiết cho xử lý bất đồng bộ.
9. **collections**: Cung cấp các loại dữ liệu Container tiên tiến.
10. **functools**: Có các chức năng bổ sung và các hoạt động trên các đối tượng có thể gọi. Hữu ích trong lập trình hàm.

# TẠO MODULE PYTHON

Tạo một module không gì khác ngoài việc lưu một đoạn mã Python bằng bất kỳ trình soạn thảo nào. Hãy lưu đoạn mã sau dưới dạng `mymodule.py`:

**< 8 Sync Dev />**

```
def SayHello(name):
   print ("Xin chào {}! Bạn có khỏe không?".format(name))
   return
```

Bạn có thể nhập `mymodule` trong phiên Python hiện tại:

**< 8 Sync Dev />**

```
>>> import mymodule
>>> mymodule.SayHello("Harish")
Xin chào Harish! Bạn có khỏe không?
```

Bạn cũng có thể nhập một module trong một tập lệnh Python khác:

**< 8 Sync Dev />**

```
import mymodule
mymodule.SayHello("Harish")
```

Chạy tập lệnh này từ dòng lệnh:

< 8 Sync Dev />

```
C:\Users\user\examples> python example.py
Xin chào Harish! Bạn có khỏe không?
```

## CÂU LỆNH IMPORT

Trong Python, câu lệnh `import` được cung cấp để tải một đối tượng Python từ một module. Đối tượng có thể là một hàm, lớp, một biến, v.v. Nếu một module chứa nhiều định nghĩa, tất cả đều sẽ được tải vào không gian tên hiện tại.

Hãy lưu đoạn mã sau có ba hàm trong `mymodule.py`:

< 8 Sync Dev />

```
def sum(x,y):
   return x+y

def average(x,y):
   return (x+y)/2

def power(x,y):
   return x**y
```

Câu lệnh `import mymodule` tải tất cả các hàm trong module này vào không gian tên hiện tại. Mỗi hàm trong module được nhập là một thuộc tính của đối tượng module này.

< 8 Sync Dev />

```
import mymodule
print ("Tổng:",mymodule.sum(10,20))
print ("Trung bình:",mymodule.average(10,20))
```

```
print

 ("Lũy thừa:",mymodule.power(10, 2))
```

Nó sẽ tạo ra đầu ra sau:

< 8 Sync Dev />

```
Tổng: 30
Trung bình: 15.0
Lũy thừa: 100
```

## CÂU LỆNH FROM ... IMPORT

Câu lệnh `import` sẽ tải tất cả các tài nguyên của module vào không gian tên hiện tại. Bạn cũng có thể nhập các đối tượng cụ thể từ một module bằng cú pháp này.

Ví dụ, nếu có ba hàm trong `mymodule` và bạn chỉ muốn nhập hai trong số chúng vào `example.py`:

< 8 Sync Dev />

```
from mymodule import sum, average
print ("Tổng:",sum(10,20))
print ("Trung bình:",average(10,20))
```

Nó sẽ tạo ra đầu ra sau:

< 8 Sync Dev />

```
Tổng: 30
Trung bình: 15.0
```

Lưu ý rằng hàm không cần phải được gọi bằng cách thêm tên của module vào trước.

## CÂU LỆNH FROM...IMPORT *

Cũng có thể nhập tất cả các tên từ một module vào không gian tên hiện tại bằng cách sử dụng câu lệnh `from...import *`. Tuy nhiên, cụm từ này nên được sử dụng cẩn thận.

## CÂU LỆNH IMPORT ... AS

Bạn có thể gán một tên alias cho module được nhập.

**< 8 Sync Dev />**

```
import mymodule as x
print ("Tổng:",x.sum(10,20))
print ("Trung bình:", x.average(10,20))
print ("Lũy thừa:", x.power(10, 2))
```

## THUỘC TÍNH CỦA MODULE

Trong Python, một module là một đối tượng của lớp module, và do đó nó được đặc trưng bởi các thuộc tính.

Dưới đây là các thuộc tính của module:

- `__file__`: Trả về tên vật lý của module.
- `__package__`: Trả về gói mà module thuộc về.
- `__doc__`: Trả về chuỗi tài liệu ở đầu module nếu có.
- `__dict__`: Trả về phạm vi toàn bộ của module.
- `__name__`: Trả về tên của module.

# VÍ DỤ

Giả sử đoạn mã sau được lưu dưới dạng `mymodule.py`:

< 8 Sync Dev />

```
"""Chuỗi tài liệu của mymodule"""
def sum(x,y):
   return x+y

def average(x,y):
   return (x+y)/2
```

Hãy kiểm tra các thuộc tính của `mymodule` bằng cách nhập nó vào đoạn mã sau:

< 8 Sync Dev />

```
import mymodule

print ("Thuộc tính __file__:", mymodule.__file__)
print ("Thuộc tính __doc__:", mymodule.__doc__)
print ("Thuộc tính __name__:", mymodule.__name__)
```

Nó sẽ tạo ra đầu ra sau:

< 8 Sync Dev />

```
Thuộc tính __file__: C:\Users\mlath\examples\mymodule.py
Thuộc tính __doc__: Chuỗi tài liệu của mymodule
Thuộc tính __name__: mymodule
```

# THUỘC TÍNH __NAME__

Thuộc tính `__name__` của một module Python có ý nghĩa quan trọng. Hãy khám phá nó chi tiết hơn.

Trong một shell tương tác, thuộc tính `__name__` trả về `'__main__'`:

```
< 8 Sync Dev />
```

```
>>> __name__
'__main__'
```

Nếu bạn nhập bất kỳ module nào trong phiên dịch viên, nó sẽ trả về tên của module là thuộc tính `__name__` của module đó.

```
< 8 Sync Dev />
```

```
>>> import math
>>> math.__name__
'math'
```

Từ bên trong một script Python, thuộc tính `__name__` trả về `'__main__'`:

```
< 8 Sync Dev />
```

```
#example.py
print ("Thuộc tính __name__ bên trong một script:", __name__)
```

Chạy điều này trong dòng lệnh:

```
< 8 Sync Dev />
```

```
Thuộc tính __name__ bên trong một script: __main__
```

Thuộc tính này cho phép một script Python được sử dụng như là một chương trình thực thi hoặc như một module. Không giống như C++, Java, C# vv, trong Python, không có khái niệm về hàm `main()`. Đoạn mã Python với phần mở rộng `.py` có thể chứa cả định nghĩa hàm cũng như các câu lệnh có thể thực thi.

Lưu `mymodule.py` và với đoạn mã sau:

```
< 8 Sync Dev />
```

```
"""Chuỗi tài liệu của mymodule"""
def sum(x,y):
   return x+y

print ("Tổng:",sum(10,20))
```

Bạn có thể thấy rằng hàm `sum()` được gọi bên trong cùng một script mà nó được định nghĩa.

```
< 8 Sync Dev />
```

```
C:\Users\user\examples> python mymodule.py
Tổng: 30
```

Bây giờ hãy nhập hàm này vào một script khác `example.py`.

```
< 8 Sync Dev />
```

```
import mymodule
print ("Tổng:",mymodule.sum(10,20))
```

Nó sẽ tạo ra đầu ra sau:

```
< 8 Sync Dev />
```

```
C:\Users\user\examples> python example.py
Tổng: 30
```

Đầu ra "Tổng:30" xuất hiện hai lần. Một lần khi module `mymodule` được nhập. Các câu lệnh thực thi trong module nhập cũng được chạy. Đầu ra thứ hai là từ script gọ i, tức là chương trình `example.py`.

Điều chúng ta muốn xảy ra là khi một module được nhập, chỉ các hàm nên được nhập, các câu lệnh thực thi của nó không được chạy. Điều này có thể được thực hiện bằng cách kiểm tra giá trị của `__name__`. Nếu nó là `__main__`, có nghĩa là nó đang chạy và không phải là import. Bao gồm các câu lệnh thực thi như cuộc gọi hàm một cách điều kiện.

Thêm câu lệnh if vào `mymodule.py` như dưới đây:

**< 8 Sync Dev />**

```
"""Chuỗi tài liệu của mymodule"""
def sum(x,y):
   return x+y

if __name__ == "__main__":
   print ("Tổng:",sum(10,20))
```

Bây giờ nếu bạn chạy chương trình `example.py`, bạn sẽ thấy rằng đầu ra "Tổng:30" chỉ xuất hiện một lần.

**< 8 Sync Dev />**

```
C:\Users\user\examples> python example.py
Tổng: 30
```

## HÀM RELOAD()

Đôi khi bạn có thể cần tải lại một module, đặc biệt là khi làm việc với phiên dịch viên tương tác của Python.

Giả sử chúng ta có một module test (test.py) với hàm sau:

< 8 Sync Dev />

```
def SayHello(name):
   print ("Xin chào {}! Bạn có khỏe không?".format(name))
   return
```

Chúng ta có thể nhập module và gọi hàm của nó từ dòng lệnh Python như sau:

< 8 Sync Dev />

```
>>> import test
>>> test.SayHello("Deepak")
Xin chào Deepak! Bạn có khỏe không?
```

Tuy nhiên, giả sử bạn cần sửa đổi hàm SayHello() như sau:

< 8 Sync Dev />

```
def SayHello(name, course):
   print ("Xin chào {}! Bạn có khỏe không?".format(name))
   print ("Chào mừng bạn đến với {} Tutorial by
8SyncDev".format(course))
   return
```

Ngay cả khi bạn chỉnh sửa tệp test.py và lưu nó, hàm được tải vào bộ nhớ sẽ không cập nhật. Bạn cần phải tải lại nó, sử dụng hàm `reload()` trong module `imp`.

< 8 Sync Dev />

```
>>> import imp
>>> imp.reload(test)
>>> test.SayHello("Deepak", "Python")
Xin chào Deepak! Bạn có khỏe không?
Chào mừng bạn đến với Python Tutorial by 8SyncDev
```