AI VIET NAM
@aivietnam.edu.vn

AI VIET NAM – AI COURSE 2025

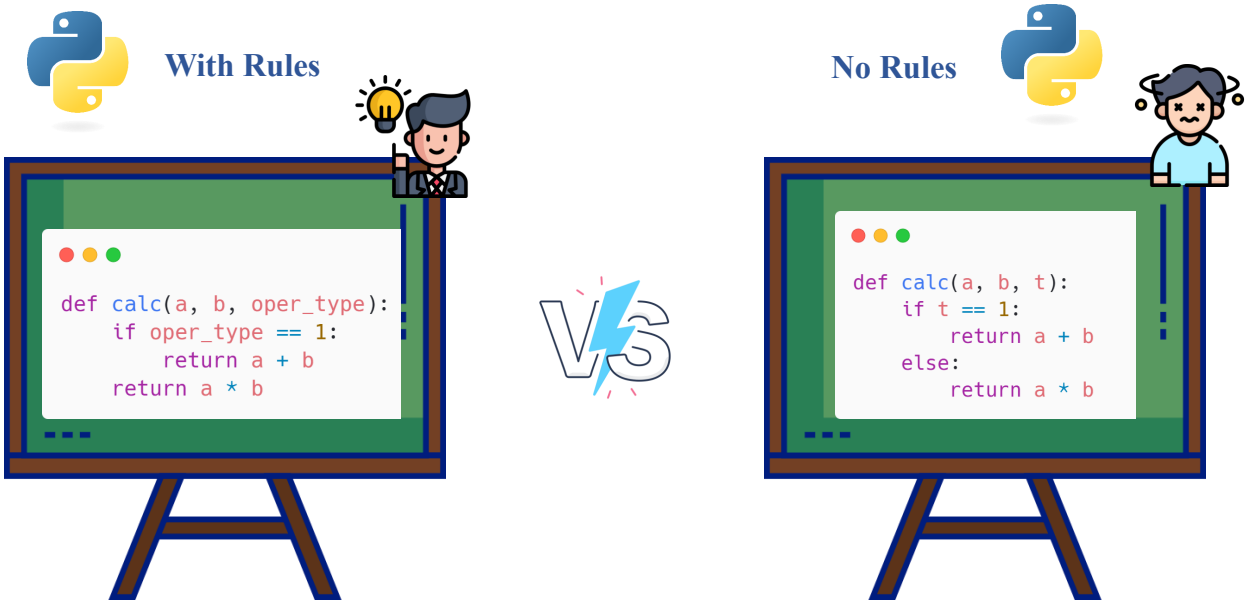
Tutorial: Naming Convention with PEP-8

Tạ Dương Anh và Nguyễn Đăng Nhã,

I. Giới thiệu

Naming Convention là một trong những nhiệm vụ quan trọng khi bắt đầu lập trình trên bất kì một ngôn ngữ nào. Các tên biến, hàm, lớp... là những thành phần xuất hiện liên tục trong code của mỗi lập trình viên và ảnh hưởng trực tiếp đến:

- Khả năng đọc hiểu code.
- Khả năng bảo trì và phát triển code.
- Khả năng cộng tác giữa các thành viên trong team.
- Giảm thiểu lỗi logic và bug.



Hình 1: So sánh giữa đặt tên có quy tác và không có quy tắc.

Dưới đây là một ví dụ sử dụng ngôn ngữ lập trình Python trong việc **đặt tên thiếu rõ ràng**:

Ví dụ đặt tên thiếu rõ ràng

```python
def calc(a, b, t):
    if t == 1:
        return a + b
    else:
        return a * b
```

Ở code trên cho thấy sự khó khăn khi đọc hiểu. Tên hàm `calc` quá chung chung, không thể hiện rõ mục đích tính toán cụ thể là gì (cộng hay nhân?). Các tham số `a`, `b` chỉ là các chữ cái, không gợi ý về ý nghĩa hay kiểu dữ liệu của chúng. Tham số `t` và giá trị 1 càng khó hiểu hơn, người đọc phải suy luận dựa vào logic bên trong mới đoán được `t` có thể là "type"(kiểu phép toán) và 1 đại diện cho phép cộng. Nếu một người khác đọc code này, họ sẽ mất thời gian để hiểu hàm làm gì và cách sử dụng nó đúng cách. Việc bảo trì hay mở rộng hàm này (ví dụ thêm phép trừ, chia) cũng sẽ trở nên phức tạp hơn.

Vậy một câu hỏi đặt ra là có một **tiêu chuẩn đặt tên** nào cho các lập trình viên cụ thể với ngôn ngữ **Python** để họ có thể tuân thủ theo từ đấy tối ưu khả năng viết code của mình không?

Câu trả lời đấy chính là **PEP8**, và nó đã trở thành tiêu chuẩn trong cộng đồng Python, giúp tạo ra một "ngôn ngữ chung"để mọi lập trình viên có thể dễ dàng làm việc với code của nhau. PEP8 (Python Enhancement Proposal 8) là tài liệu hướng dẫn về phong cách lập trình Python, được viết bởi Guido van Rossum (cha đẻ của Python), Barry Warsaw và Nick Coghlan. Đây là quy chuẩn chính thức về cách viết code Python, bao gồm:

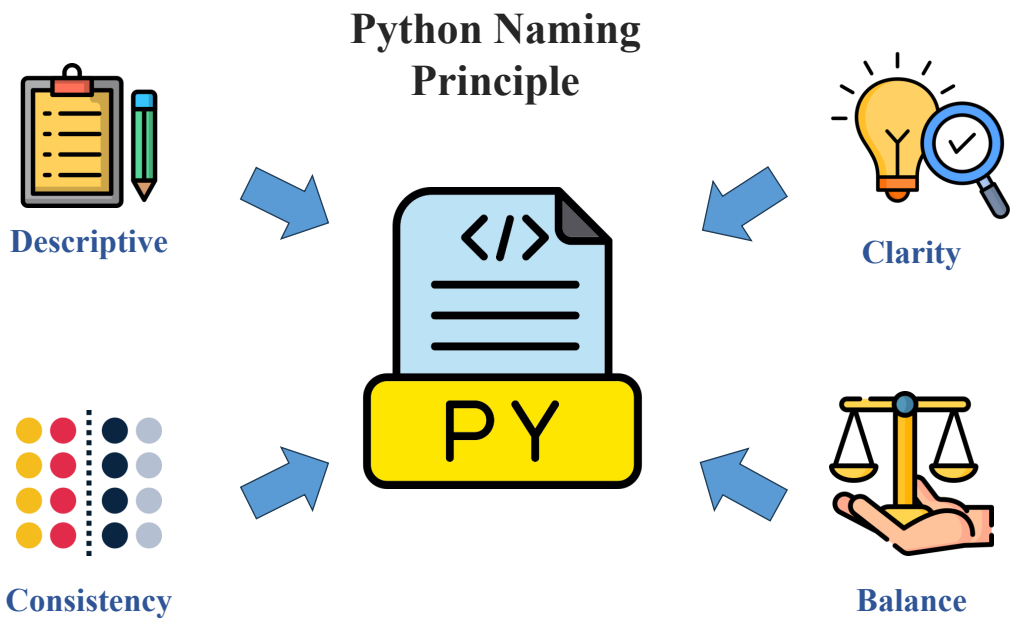
- Quy ước đặt tên.
- Định dạng code.
- Import cách sử dụng các module
- Comment và documentation

Mục lục

II. Nguyên tắc đặt tên trong Python

II.1. Nguyên tắc chung

Khi lập trình Python, việc đặt tên đúng cách không chỉ giúp mã nguồn dễ đọc và dễ bảo trì mà còn thể hiện sự chuyên nghiệp trong phát triển phần mềm. Một tên biến, hàm hay lớp được đặt tốt sẽ giúp người khác (và chính chúng ta trong tương lai) nhanh chóng hiểu được mục đích của nó mà không cần đọc hết toàn bộ logic bên trong. Để đảm bảo điều này, việc đặt tên nên tuân theo một số nguyên tắc cơ bản: tính mô tả, tính nhất quán, tính rõ ràng, và tính ngắn gọn vừa phải. Những nguyên tắc này không chỉ giúp tăng chất lượng mã nguồn mà còn góp phần làm cho dự án trở nên dễ hiểu, dễ mở rộng và dễ cộng tác hơn.



Hình 2: 4 nguyên tắc đặt tên chung.

1. **Tính mô tả (Descriptive)**: Tên nên mô tả chính xác mục đích hoặc chức năng của đối tượng.
   - **Tốt:** `customer_age`, `calculate_total_price`
   - **Không tốt:** `a`, `do_stuff`
2. **Tính nhất quán (Consistency)**: Áp dụng cùng một quy ước đặt tên trong toàn bộ dự án.
   - Không nên trộn lẫn các kiểu đặt tên như `getUserName` và `get_user_email` trong cùng một dự án.

3. **Tính rõ ràng (Clarity)**: Tên nên rõ ràng và tránh viết tắt khó hiểu.
   - **Tốt:** `calculate_average_score`
   - **Không tốt:** `calc_avg_scr`
4. **Tính ngắn gọn vừa phải**: Tên nên đủ ngắn để dễ đọc nhưng đủ dài để mô tả chức năng.
   - **Tốt:** `user_login_count`
   - **Không tốt:** `number_of_times_a_user_has_logged_into_the_system`

II.2. Tránh sử dụng tên biến đơn ký tự

Python khuyến nghị tránh sử dụng một số ký tự đơn lẻ vì chúng dễ nhầm lẫn:

- l (chữ L thường): Dễ nhầm với số 1
- O (chữ O hoa): Dễ nhầm với số 0
- I (chữ I hoa): Dễ nhầm với số 1 hoặc chữ l thường

Ví dụ về trường hợp dễ gây nhầm lẫn:

Các chữ cái dễ nhầm lẫn

```
O = 1    # Nhìn như số 0 = 1
l = 10   # Nhìn như số 1 = 10
I = 100  # Nhìn như số 1 = 100
```

II.3. Giới hạn bộ ký tự và khuyến nghị

Python cho phép sử dụng nhiều ký tự khác nhau trong tên, nhưng bạn nên:

1. Sử dụng các ký tự ASCII khi có thể
2. Chỉ sử dụng chữ cái (a-z, A-Z), số (0-9) và dấu gạch dưới (_)
3. Bắt đầu bằng chữ cái hoặc dấu gạch dưới (_), không bắt đầu bằng số
4. Phân biệt chữ hoa và chữ thường (case-sensitive)

III. Quy ước đặt tên theo loại

III.1. Biến

Trong Python, **biến (Variables)** sử dụng quy ước chữ thường với dấu gạch dưới (snake_case):

Ví dụ về đặt tên biến (snake_case)

```
user_name = "Nguyen Van A"
total_items = 50
is_active = True
```

Quy tắc đặt tên biến cần lưu ý ở đây là:

- Sử dụng chữ thường
- Ngăn cách các từ bằng dấu gạch dưới
- Tên nên mô tả giá trị mà biến chứa
- Tên biến boolean nên bắt đầu bằng `is_`, `has_`, `can_`, `should_`...

III.2. Hàm

Hàm (Functions) cũng sử dụng quy ước snake_case giống như biến:

Ví dụ về đặt tên hàm (snake_case)

```
def calculate_total_price(price, quantity, discount=0):
    return price * quantity * (1 - discount)
```

Quy tắc đặt tên hàm cần lưu ý ở đây là:

- Sử dụng động từ mô tả hành động: `get_`, `calculate_`, `process_`, `validate_`...
- Hàm trả về boolean nên bắt đầu bằng: `is_`, `has_`, `can_`, `should_`...
- Tên nên mô tả chức năng, không mô tả cách thực hiện

III.3. Lớp

Lớp (Classes) sử dụng quy ước CapWords/PascalCase (mỗi từ viết hoa chữ cái đầu, không có gạch dưới):

Ví dụ về đặt tên lớp (CapWords/PascalCase)

```
class UserAccount:
    def __init__(self, username, email):
        self.username = username
        self.email = email
```

Quy tắc đặt tên hàm cần lưu ý ở đây là:

- Bắt đầu bằng chữ hoa
- Mỗi từ bắt đầu bằng chữ hoa
- Không sử dụng dấu gạch dưới giữa các từ
- Tên nên là danh từ hoặc cụm danh từ
- Đặt tên theo đối tượng mà lớp đại diện

Lưu ý: Lớp có liên quan đến các thư viện khác có thể tuân theo quy ước riêng của thư viện đó.

III.4. Phương thức

Phương thức (Methods) trong lớp sử dụng quy ước snake_case giống như hàm:

Ví dụ về đặt tên phương thức (snake_case)

```
class Customer:
    def __init__(self, name, email):
        self.name = name
        self.email = email

    def get_full_info(self):
        return f"Name: {self.name}, Email: {self.email}"

    def update_email(self, new_email):
        self.email = new_email
```

Phương thức đặc biệt (Special method names): Python có các phương thức đặc biệt bắt đầu và kết thúc bằng hai dấu gạch dưới (Dunder methods):

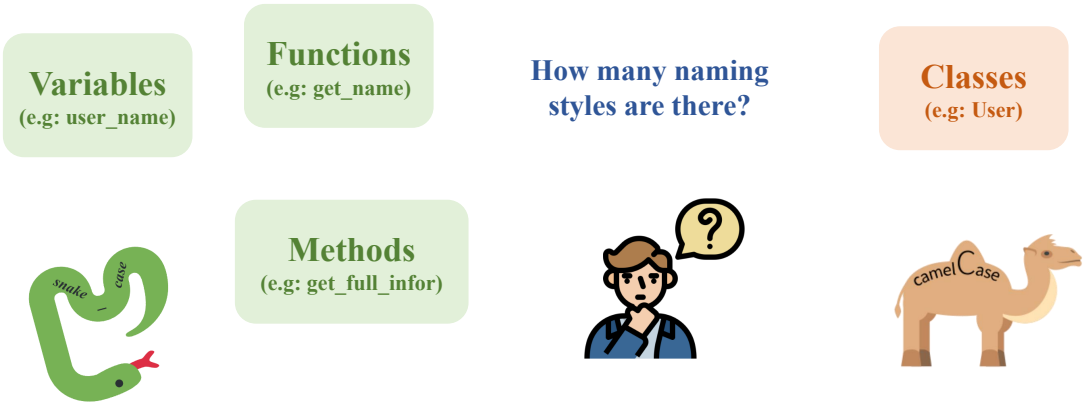
Ví dụ về phương thức đặc biệt (Dunder methods)

```python
class Product:
    def __init__(self, name, price):  # Constructor
        self.name = name
        self.price = price

    def __str__(self):  # String representation
        return f"{self.name} - {self.price} VND"

    def __eq__(self, other):  # Equality comparison
        return self.price == other.price
```

Ở **Hình 3** này sẽ giúp chúng ta có cái nhìn tổng quát hơn về các quy ước đặt tên theo loại (Variables, Functions, Methods, Classes):



Hình 3: Quy ước đặt tên theo loại (Variables, Functions, Methods, Classes)

III.5. Hằng số

Hằng số (Constants) sử dụng quy ước chữ HOA với dấu gạch dưới (UPPER_CASE_WITH_UNDERSCORES):

Ví dụ về đặt tên hằng số (UPPER_CASE_WITH_UNDERSCORES)

```python
PI = 3.14159
MAX_CONNECTIONS = 100
DEFAULT_TIMEOUT = 30
```

```
DATABASE_URL = "postgresql://user:password@localhost/dbname"
```

Quy tắc đặt tên hằng số cần lưu ý ở đây là:

- Giá trị không thay đổi trong suốt quá trình chạy chương trình
- Giá trị có ý nghĩa đặc biệt hoặc được sử dụng nhiều lần
- Cấu hình chương trình
- Các giá trị magic number cần được đặt tên rõ ràng

III.6. Module & Gói

Module: Module sử dụng tên ngắn, chữ thường:

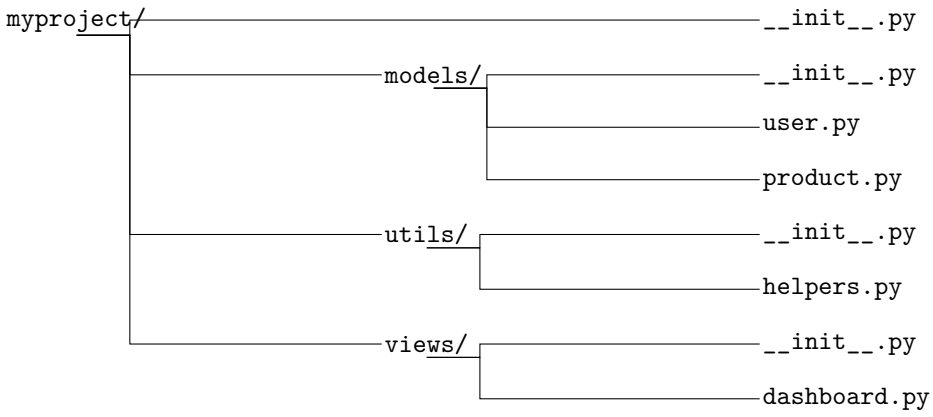
Ví dụ về đặt tên module (user_management.py)

```python
# file: user_management.py
def get_user(user_id):
    pass
def create_user(user_data):
    pass
```

Quy tắc đặt tên Module cần lưu ý ở đây là:

- Tên ngắn, rõ ràng
- Chữ thường
- Có thể sử dụng dấu gạch dưới nếu cần thiết để tăng tính đọc hiểu
- Tránh dùng tên trùng với module chuẩn của Python (như string, sys)

Gói (Packages): Gói sử dụng tên ngắn, chữ thường và tránh sử dụng dấu gạch dưới:



Quy tắc đặt tên Gói cần lưu ý ở đây là:

- Tên ngắn, rõ ràng
- Chữ thường
- Tránh sử dụng dấu gạch dưới
- Tránh trùng với tên gói đã có trên PyPI

IV. Mẫu đặt tên nâng cao

IV.1. Thuộc tính/Phương thức riêng tư, công khai và bảo vệ

Trong Python, quyền truy cập vào thuộc tính và phương thức chủ yếu được quản lý thông qua quy ước đặt tên và một cơ chế đặc biệt gọi là mã hóa tên (name mangling), chứ không phải bằng các từ khóa bắt buộc như private hay protected trong các ngôn ngữ như Java hoặc C++.

Thuộc tính/Phương thức Công khai (Public) Không có dấu gạch dưới ở đầu, có thể truy cập từ bất kỳ đâu:

Ví dụ về thuộc tính và phương thức Công khai (Public)

```
class Example:
    def __init__(self, data):
        self.public_attribute = data  # Thuộc tính Public

    def public_method(self):      # Phương thức Public
        print(f"Public data: {self.public_attribute}")

# Truy cập từ bên ngoài
obj = Example("Hello")
print(obj.public_attribute)  # Hợp lệ
obj.public_method()          # Hợp lệ
```

Thuộc tính/Phương thức riêng tư (Private) Bắt đầu bằng hai dấu gạch dưới đơn (___), thể hiện rằng chỉ có thể truy cập từ bên trong chính lớp đó. Không thể truy cập từ bên ngoài hoặc từ các lớp con.

Ví dụ về thuộc tính và phương thức Riêng tư (Private)

```
class MyClass:
    def __init__(self):
        self.__private_attribute = "Tôi là Private (bị mã hóa tên)" # Thuộc tính
                                                                    Private

    def public_method_accessing_private(self): # Phương thức Public
        print(f"Accessing private from inside: {self.__private_attribute}") # Có thể
```

```
                                        truy cập từ bên trong lớp

    def __private_method(self): # Phương thức Private (bị mã hóa tên)
        print("Đây là phương thức Private.")

    def public_method_calling_private_method(self): # Phương thức Public
        self.__private_method() # Có thể gọi từ bên trong lớp

# Truy cập từ bên ngoài
obj = MyClass()

obj.public_method_accessing_private() # Hợp lệ

# print(obj.__private_attribute) # Sẽ gây lỗi: AttributeError
# obj.__private_method()       # Sẽ gây lỗi: AttributeError

# Truy cập Private thông qua tên đã bị mã hóa (không khuyến khích)
# print(obj._MyClass__private_attribute) # Có thể, nếu biết tên mã hóa

obj.public_method_calling_private_method() # Cách đúng để tương tác là thông qua phương
                                            thức public
```

Lưu ý: Bạn vẫn có thể truy cập thành viên bị mã hóa tên từ bên ngoài nếu biết tên đã bị mã hóa, nhưng điều này là bất thường và không khuyến khích.

Thuộc tính/Phương thức bảo vệ (Protected) Python không có khái niệm "protected"như các ngôn ngữ lập trình khác, nhưng quy ước dùng một dấu gạch dưới (_) thường được hiểu là thuộc tính "protected có thể truy cập từ lớp và lớp con, nhưng không nên truy cập từ bên ngoài.

Ví dụ về thuộc tính và phương thức Bảo vệ (Protected)

```
class Base:
    def __init__(self):
        self._protected_attribute = "Tôi là Protected (theo quy ước)" # Thuộc tính
                                            Protected (theo quy ước)

    def _protected_method(self): # Phương thức Protected (theo quy ước)
        print(self._protected_attribute)

class Derived(Base): # Lớp con kế thừa
    def access_protected(self):
        # Lớp con có thể truy cập thành viên protected của lớp cha (theo quy ước)
        self._protected_method()
        print(f"Accessed from Derived: {self._protected_attribute}")

# Truy cập từ bên ngoài
base_obj = Base()
# print(base_obj._protected_attribute) # Có thể truy cập, nhưng không nên theo quy ước

derived_obj = Derived()
derived_obj.access_protected() # Hợp lệ theo quy ước (lớp con truy cập)
```

Lưu ý: Đây là một quy ước cho lập trình viên khác biết rằng thành viên này được thiết kế cho mục đích sử dụng nội bộ hoặc cho các lớp con. Python không ngăn chặn việc truy cập thành viên này từ bên ngoài, nhưng làm như vậy được coi là vi phạm quy ước và không nên.

IV.2. Phương thức đặc biệt

Dunder methods (viết tắt của "double underscore") là các phương thức đặc biệt trong Python, bắt đầu và kết thúc bằng hai dấu gạch dưới:

Ví dụ về triển khai Dunder Methods

```python
class Product:
    def __init__(self, name, price):
        self.name = name
        self.price = price

    def __str__(self):
        return f"{self.name} (\${self.price})"

    def __repr__(self):
        return f"Product('{self.name}', {self.price})"

    def __eq__(self, other):
        if not isinstance(other, Product):
            return False
        return self.name == other.name and self.price == other.price
```

Dunder methods phổ biến:

1. `__init__(self, ...)`: Constructor, khởi tạo đối tượng
2. `__str__(self)`: Chuyển đối tượng thành chuỗi thân thiện với người dùng
3. `__repr__(self)`: Biểu diễn lập trình của đối tượng
4. `__len__(self)`: Trả về độ dài của đối tượng
5. `__getitem__(self, key)`: Cho phép truy cập kiểu chỉ mục `obj[key]`
6. `__eq__(self, other)`: So sánh đối tượng

V. Đặt tên theo ngữ cảnh cụ thể

V.1. Biến vòng lặp và bộ lặp

Trong các vòng lặp ngắn, thường sử dụng tên biến ngắn gọn, nhưng vẫn nên mô tả đúng mục đích:

Ví dụ về đặt tên biến trong vòng lặp

```python
# Vòng lặp đơn giản
for i in range(10):
    print(i)

# Vòng lặp với nhiều phần tử
for user in users:
    print(user.name)

# Với enumeration
for idx, value in enumerate(values):
    print(f"Phần tử thứ {idx}: {value}")

# Với dictionary
for key, value in data.items():
    print(f"{key}: {value}")
```

Quy tắc đặt tên biến vòng lặp:

- Sử dụng `i`, `j`, `k` cho các index đơn giản
- Chữ thường
- Đối với vòng lặp qua các đối tượng, sử dụng tên mô tả dạng số ít của loại đối tượng: `user` (thay vì `users`), `product` (thay vì `products`)
- Tránh sử dụng tên quá chung chung như `data`, `item`, `thing`

V.2. Tên ngoại lệ

Lớp ngoại lệ tùy chỉnh nên theo quy ước tên lớp (PascalCase) và kết thúc bằng "Error"hoặc "Exception":

Ví dụ về đặt tên lớp ngoại lệ tùy chỉnh

```python
class InvalidUserError(Exception):
    pass

class DatabaseConnectionError(Exception):
    pass

```

```python
class ConfigurationException(Exception):
    pass

try:
    # Some code
    if not valid_user:
        raise InvalidUserError("User information is invalid")
except InvalidUserError as e:
    print(f"Error: {e}")
```

V.3. Context Managers

Context managers (sử dụng với `with` statement) nên được đặt tên mô tả hành động hoặc tài nguyên được quản lý:

Ví dụ về Context Manager sử dụng lớp

```python
class DatabaseConnection:
    def __enter__(self):
        # Khởi tạo kết nối
        self.connection = connect_to_db()
        return self.connection
    def __exit__(self, exc_type, exc_val, exc_tb):
        # Đóng kết nối
        self.connection.close()
# Sử dụng
with DatabaseConnection() as conn:
    results = conn.execute("SELECT * FROM users")
```

Hoặc với hàm và decorator `@contextmanager`:

Ví dụ về Context Manager sử dụng hàm và decorator

```python
from contextlib import contextmanager

@contextmanager
def open_file(filename, mode="r"):
    file = open(filename, mode)
    try:
        yield file
    finally:
        file.close()

# Sử dụng
with open_file("data.txt") as f:
    content = f.read()
```

V.4. Tham số và đối số hàm

Tham số hàm nên được đặt tên rõ ràng để người sử dụng hàm có thể hiểu mà không cần đọc tài liệu:

Ví dụ về đặt tên tham số hàm

```python
# Tốt
def create_user(username, email, role="user", is_active=True):
    # ...

# Chưa tốt
def create_user(u, e, r="user", a=True):
    # ...
```

- Tham số đầu tiên của phương thức trong lớp luôn là `self`
- Tham số đầu tiên của phương thức lớp (class method) luôn là `cls`
- Tham số có giá trị mặc định nên được đặt sau các tham số bắt buộc
- Tham số kiểu * `args` và **`kwargs` nên giữ nguyên tên này khi được sử dụng cho mục đích truyền tiếp đối số

V.5. Type Hints và đặt tên

Khi sử dụng type hints, tên biến có thể được chọn để phản ánh kiểu dữ liệu:

Ví dụ về sử dụng Type Hints

```python
from typing import List, Dict, Optional, Tuple

# Tốt
def process_user_data(user_id: int, settings: Dict[str, str]) -> Optional[User]:
    # ...

# Chưa tốt
def process_data(id: int, s: Dict[str, str]) -> Optional[User]:
    # ...
```

Tuy nhiên, không nên sử dụng cách đặt tên kiểu Hungarian notation (bao gồm kiểu dữ liệu trong tên):

Ví dụ về cách đặt tên không nên dùng (Hungarian Notation

```python
# Không nên
str_name = "John"  # Không nên bao gồm kiểu dữ liệu (str) trong tên
int_count = 5      # Không nên bao gồm kiểu dữ liệu (int) trong tên

```

```python
# Nên
name = "John"
count = 5
```

VI. Lỗi thường gặp và cách tránh

VI.1. Tên quá viết tắt

Tên biến, hàm quá ngắn và viết tắt quá mức có thể làm giảm khả năng đọc hiểu code:

Code so sánh tên viết tắt và tên rõ ràng

```python
# Tên quá ngắn và viết tắt quá mức
def clc_tx(p, q, r):
    return p * q * r / 100

# Tên rõ ràng, dễ hiểu
def calculate_tax(price, quantity, rate):
    return price * quantity * rate / 100
```

Làm thế nào để tránh:

- Đặt tên đầy đủ, rõ ràng
- Chỉ sử dụng viết tắt phổ biến và được hiểu rộng rãi (như id, max, min)
- Nếu cần viết tắt, hãy đảm bảo tính nhất quán trong toàn bộ dự án

VI.2. Đặt tên chưa cụ thể

Tên quá chung chung không cung cấp đủ thông tin về mục đích hoặc chức năng:

Code so sánh tên chung chung và tên cụ thể

```python
# Tên quá chung chung
data = get_data()
result = process(data)
save(result)

# Tên cụ thể, rõ ràng
user_profiles = get_user_profiles()
filtered_profiles = filter_active_users(user_profiles)
save_to_database(filtered_profiles)
```

Làm thế nào để tránh:

- Đặt tên mô tả cụ thể nội dung hoặc mục đích
- Bao gồm thông tin về ngữ cảnh khi cần thiết

VI.3. Phong cách đặt tên không nhất quán

Trộn lẫn các phong cách đặt tên khác nhau làm giảm tính nhất quán và khó đọc:

Code so sánh phong cách không nhất quán và nhất quán

```python
# Phong cách không nhất quán
class UserAccount:
    def __init__(self, userName, email_address):
        self.UserName = userName
        self.emailAddress = email_address

    def GetFullName(self):
        return self.UserName

    def update_email(self, new_email):
        self.emailAddress = new_email

# Phong cách nhất quán
class UserAccount:
    def __init__(self, user_name, email_address):
        self.user_name = user_name
        self.email_address = email_address

    def get_full_name(self):
        return self.user_name

    def update_email(self, new_email):
        self.email_address = new_email
```

Làm thế nào để tránh:

- Thống nhất một phong cách đặt tên trong toàn bộ dự án
- Tuân thủ PEP8 và quy ước chung của Python
- Sử dụng công cụ linting để phát hiện và sửa lỗi nhất quán

VI.4. Sử dụng từ khóa đã được dành riêng

Python có một số từ khóa được dành riêng không thể sử dụng làm tên biến, hàm hoặc lớp:

Ví dụ về việc sử dụng từ khóa và tên built-in

```python
# Lỗi: sử dụng từ khóa
class = "Advanced"  # Lỗi: 'class' là từ khóa
def = 42            # Lỗi: 'def' là từ khóa
```

```

# Lỗi: sử dụng tên giống với hàm built-in
list = [1, 2, 3]    # Không gây lỗi nhưng ghi đè lên hàm list() built-in
sum = 10            # Ghi đè lên hàm sum() built-in
```

Từ khóa phổ biến trong Python: False, None, True, and, as, assert, async, await, break, class, continue, def, del, elif, else, except, finally, for, from, global, if, import, in, is, lambda, nonlocal, not, or, pass, raise, return, try, while, with, yield

Làm thế nào để tránh:

- Không đặt tên trùng với từ khóa Python
- Nếu cần đặt tên gần với từ khóa, có thể thêm dấu gạch dưới: class_ thay vì class

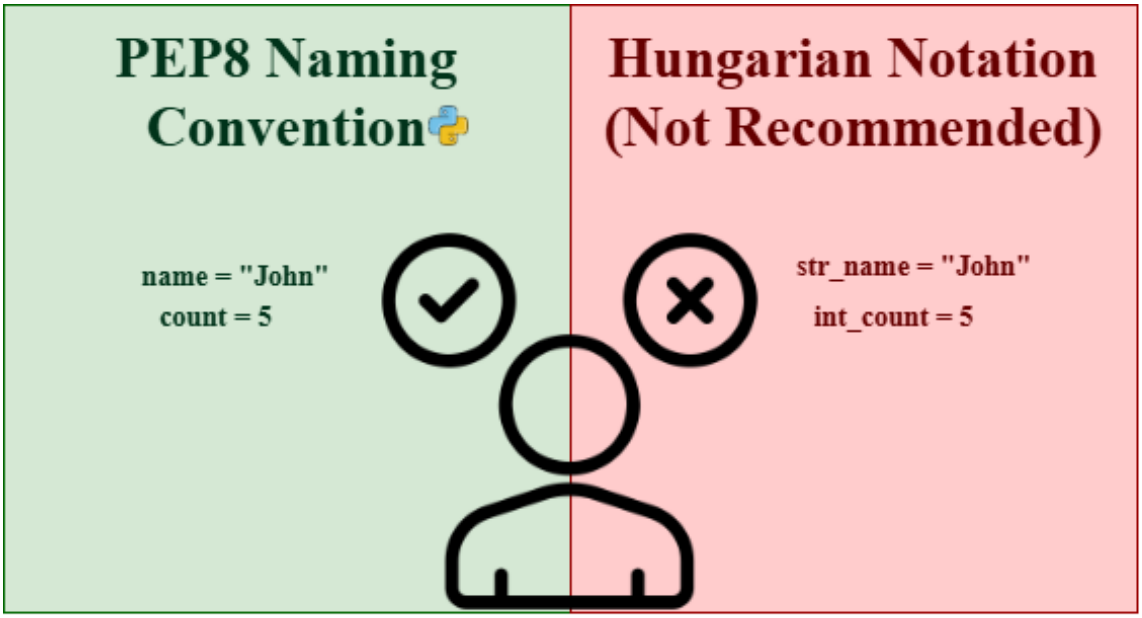
Ví dụ về cách xử lý khi tên trùng từ khóa

```
# Cách xử lý khi cần dùng tên gần với từ khóa
class_ = "Advanced"   # Thêm dấu gạch dưới
list_ = [1, 2, 3]     # Thêm dấu gạch dưới
```

VI.5. Hungarian Notation và lý do không khuyến khích trong Python

Hungarian Notation là phong cách đặt tên trong đó kiểu dữ liệu được đặt ở đầu tên biến. Ví dụ: `strName`, `intCount`, `bIsActive`. Phong cách này không được khuyến khích trong Python vì:

- Python là ngôn ngữ kiểu động, kiểu dữ liệu có thể thay đổi
- Type hints đã hỗ trợ việc chú thích kiểu
- Làm cho tên biến dài và khó đọc hơn
- Không phù hợp với triết lý "rõ ràng hơn là ngầm định"của Python



Hình 4: PEP8 Naming Convention vs Hungarian Notation

VII. Một số công cụ giúp cải thiện naming convention

VII.1. Linters kiểm tra tuân thủ quy ước đặt tên

Python có nhiều công cụ linting giúp kiểm tra tự động việc tuân thủ quy ước đặt tên:

1. **pylint**: Công cụ phân tích mã nguồn Python mạnh mẽ, kiểm tra cả quy ước đặt tên và nhiều lỗi khác
2. **flake8**: Công cụ kết hợp PyFlakes, pycodestyle và Ned Batchelder's McCabe script
3. **pycodestyle** (trước đây gọi là pep8): Kiểm tra code theo quy ước PEP8

VII.2. Công cụ tái cấu trúc tự động

1. **Rope**: Thư viện tái cấu trúc Python, có thể đổi tên biến, hàm, lớp đồng bộ trong toàn bộ dự án
2. **PyCharm Refactoring**: Cung cấp tính năng đổi tên an toàn (Shift+F6), phát hiện và thay đổi tất cả các vị trí sử dụng
3. **black**: Công cụ định dạng code tự động, không thay đổi tên nhưng làm cho code nhất quán về định dạng

VIII. Khi nào nên phá vỡ quy ước

Mặc dù tuân thủ quy ước là quan trọng, nhưng có một số trường hợp hợp lý để phá vỡ quy ước:
Tương thích với code hiện có: Khi làm việc với một dự án đã có sẵn sử dụng quy ước đặt tên khác với PEP8, ưu tiên sự nhất quán trong dự án hơn là áp dụng máy móc PEP8:

Ví dụ giữ tính nhất quán với code hiện có

```
# Nếu dự án sử dụng camelCase, tiếp tục sử dụng nó
def getUserData(userId):
    # ...

def updateUserProfile(userProfile):
    # ...
```

Tích hợp với thư viện: Khi làm việc với thư viện bên ngoài có quy ước riêng, có thể cần tuân theo quy ước của thư viện đó ở phần code tương tác với nó:

Ví dụ tích hợp với quy ước của thư viện (Django)

```
# Ví dụ với Django, sử dụng snake_case cho phương thức
class UserModel(models.Model):
    first_name = models.CharField(max_length=100)
    last_name = models.CharField(max_length=100)

    def get_full_name(self):  # snake_case theo quy ước Django
        return f"{self.first_name} {self.last_name}"

# Ví dụ với Qt, sử dụng camelCase cho slot
class MyWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.button.clicked.connect(self.onButtonClick)  # camelCase cho slot Qt

    def onButtonClick(self):  # camelCase cho slot Qt
        # ...
```

Yêu cầu của framework: Một số framework yêu cầu tên cụ thể cho các hook hoặc callback, buộc phải tuân theo quy ước của framework:

Ví dụ tích hợp với quy ước của thư viện (Qt)

```
# Flask routes với dấu gạch ngang trong URL
@app.route('/user-profile/<user_id>')
def user_profile(user_id):
    # ...

# Django model với phương thức đặc biệt
class Article(models.Model):
```

```
    title = models.CharField(max_length=200)

    def save(self, *args, **kwargs):  # Tên phương thức theo yêu cầu của Django
        # ...
        super().save(*args, **kwargs)
```

Lưu ý quan trọng: Khi phá vỡ quy ước, hãy:

1. Có lý do chính đáng
2. Đảm bảo nhất quán trong phạm vi áp dụng
3. Ghi chú rõ ràng lý do trong code hoặc tài liệu
4. Tránh trộn lẫn nhiều quy ước khác nhau

- Hết -