**THỰC HÀNH BUỔI 2**

# Mục tiêu

* Tìm hiểu Python cơ bản, các kiểu dữ liệu: List, Dict, Set, Tuple
* Làm các bài tập chuyển đổi dữ liệu, vòng lặp, xử lý điều kiện.
* Thực hành viết hàm xử lý dữ liệu (thêm, sửa, xóa trong list).
* Bắt đầu lập trình lưu “task” vào list.

# Lý thuyết

## Python - Ngôn ngữ lập trình nguồn mở đa năng

**Python là gì và tại sao nên học?**

* Python là ngôn ngữ lập trình mã nguồn mở, được phát triển bởi Guido van Rossum từ năm 1991
* Triết lý thiết kế: "Simple is better than complex" - Đơn giản hơn là tốt hơn phức tạp
* Cú pháp gần gũi với ngôn ngữ tự nhiên, dễ đọc, dễ hiểu cho người mới bắt đầu.

## Ứng dụng thực tế của Python:

* **Web Development**: Django, Flask, FastAPI
* **Data Science & AI**: NumPy, Pandas, TensorFlow, PyTorch
* **Automation & Scripting**: Tự động hóa tác vụ, xử lý file
* **Desktop Applications**: Tkinter, PyQt
* **Game Development**: Pygame

# Clean Code là gì?

Link tham khảo: <https://peps.python.org/pep-0008/>

Clean code là mã nguồn rõ ràng, dễ đọc, dễ bảo trì, có khả năng mở rộng và dễ dàng kiểm thử. Nó được viết theo cách mà người khác (đồng nghiệp) hoặc chính bạn trong tương lai có thể dễ dàng hiểu, cải tiến hoặc bảo trì.

A diagram of a clean code

AI-generated content may be incorrect.

## Tầm Quan Trọng Của Clean Code

Clean Code không chỉ là phong cách viết code đơn thuần mà còn là yếu tố quan trọng giúp tăng hiệu suất làm việc, cải thiện hợp tác nhóm và giảm chi phí bảo trì dài hạn.

## Nguyên tắc PEP8 là gì?

**PEP8** (Python Enhancement Proposal 8) là bộ quy chuẩn coding style được khuyến nghị cho tất cả lập trình viên Python, nhằm giúp mã nguồn dễ đọc, dễ hiểu, thống nhất khi làm việc nhóm hoặc cộng đồng.

* **Mục tiêu:**
  + Mã nguồn nhất quán, dễ bảo trì, tránh lỗi vặt do trình bày khác nhau.
  + Được áp dụng từ các dự án nhỏ đến lớn, mã nguồn mở hoặc sản phẩm thực tế.

## Một số quy tắc PEP8 quan trọng

### Indentation (Thụt lề)

* Luôn dùng 4 dấu cách (spaces), KHÔNG dùng tab.
* Các khối code (hàm, vòng lặp, điều kiện) thụt vào 4 spaces.

### Đặt tên biến, hàm

* **Biến/hàm**: Sử dụng kiểu snake\_case (chữ thường, gạch dưới):  
  my\_variable, process\_data()
* A screenshot of a computer

  AI-generated content may be incorrect.
* **Class**: Sử dụng kiểu CamelCase:  
  MyClass, StudentInfo

A close-up of a math equation

AI-generated content may be incorrect.

* **Hằng số**: Chữ in hoa, gạch dưới:  
  MAX\_SIZE, DEFAULT\_COLOR

**A computer screen shot of a math problem

AI-generated content may be incorrect.**

### Độ dài dòng

A close-up of a paper

AI-generated content may be incorrect.

### Khoảng trắng

* Không thêm khoảng trắng dư thừa:
  + Không có khoảng trắng bên trong dấu ngoặc:  
    my\_list = [1, 2, 3]  
    Sai: my\_list = [ 1, 2, 3 ]
  + Sau dấu phẩy, thêm 1 space:  
    func(a, b)
  + Giữa các phép toán:  
    x = a + b

### Dấu nháy

* Có thể dùng cả dấu nháy đơn '...' và nháy kép "...". Tuy nhiên, nhất quán trong dự án.

### Comment (Chú thích)

* Comment code khi có logic phức tạp.
* Docstring không chỉ giúp người đọc hiểu code mà còn hỗ trợ tích hợp với các công cụ như Sphinx để tạo tài liệu tự động và IDE để cung cấp gợi ý thông minh  
  Sử dụng tiếng Việt chuẩn hoặc tiếng Anh.  
  Ví dụ:

# Hàm tính điểm trung bình từng môn

### Type Annotations

Type annotations giúp kiểm tra kiểu dữ liệu, phát hiện lỗi sớm và tối ưu trải nghiệm IDE.

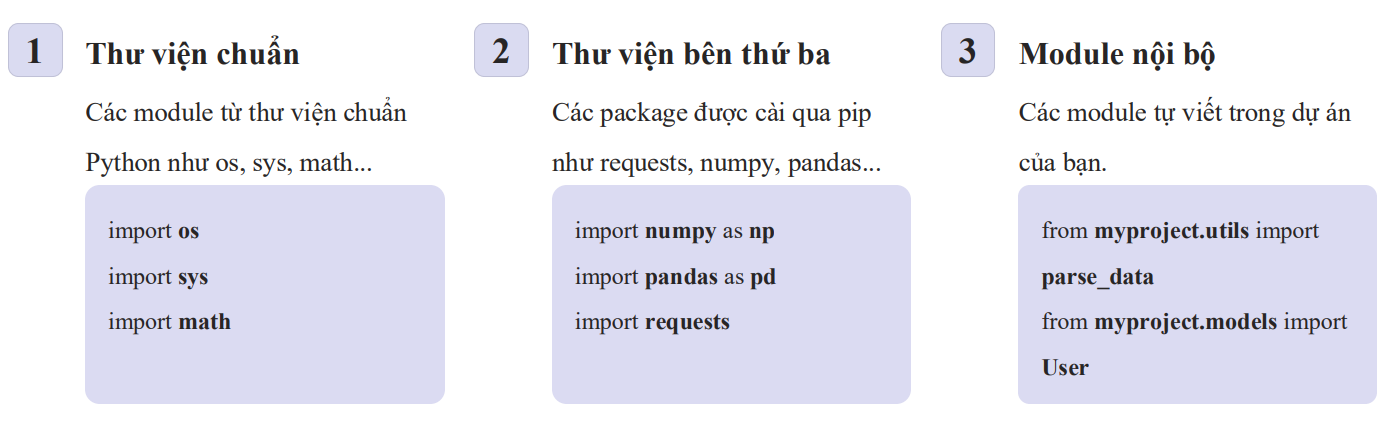


### Các dòng trống

* Giữa các hàm/class nên có tối thiểu 2 dòng trống để tăng độ rõ ràng.

### Import

* Tất cả các lệnh import nên ở đầu file.



### Áp dụng thực tế PEP8

* Sử dụng công cụ hỗ trợ như **flake8**, **black**, **pylint** để kiểm tra và tự động sửa style code.
* Khi review/practice, kiểm tra lại code theo checklist PEP8.

## Cấu trúc dữ liệu cơ bản trong Python

### List (Danh sách)

* Là tập hợp có thứ tự, có thể thay đổi (mutable)
* Cho phép lưu trữ nhiều kiểu dữ liệu khác nhau
* Sử dụng dấu ngoặc vuông []

| **Phương thức** | **Mô tả** | **Ví dụ sử dụng** |
| --- | --- | --- |
| append(x) | Thêm phần tử x vào cuối list | lst.append(10) |
| extend(iterable) | Mở rộng list bằng các phần tử từ iterable | lst.extend([1,2,3]) |
| insert(i, x) | Chèn phần tử x vào vị trí index i | lst.insert(0, 'a') |
| remove(x) | Xóa phần tử đầu tiên có giá trị x | lst.remove(3) |
| pop([i]) | Lấy và xóa phần tử vị trí i (mặc định cuối) | lst.pop() hoặc lst.pop(1) |
| clear() | Xóa hết tất cả phần tử trong list | lst.clear() |
| index([start, end]) | Trả về chỉ số đầu tiên của phần tử x trong list | lst.index(2) |
| count(x) | Đếm số lần xuất hiện của phần tử x | lst.count(5) |
| sort(key=None, reverse=False) | Sắp xếp list tại chỗ | lst.sort()  hoặc lst.sort(reverse=True) |
| reverse() | Đảo ngược vị trí các phần tử trong list tại chỗ | lst.reverse() |
| copy() | Trả về bản sao (shallow copy) của list | new\_list = lst.copy() |

SlicingA screenshot of a computer

AI-generated content may be incorrect.

lst = [3, 1, 4]

lst.append(2) # lst = [3, 1, 4, 2]

lst.insert(1, 5) # lst = [3, 5, 1, 4, 2]

lst.remove(4) # lst = [3, 5, 1, 2]

print(lst.pop()) # Output: 2, lst = [3, 5, 1]

lst.sort() # lst = [1, 3, 5]

lst.reverse() # lst = [5, 3, 1]

print(lst.count(3)) # Output: 1

### Tuple (Bộ)

* Tương tự **List** nhưng **không thể thay đổi (immutable)**
* Sử dụng dấu ngoặc tròn ()
* Thường dùng để **lưu trữ dữ liệu không đổi như tọa độ, thông tin cá nhân**

| **Phương thức** | **Mô tả** | **Ví dụ sử dụng** |
| --- | --- | --- |
| count(x) | Đếm số lần xuất hiện của phần tử x trong tuple | (1,2,2,3).count(2) # 2 |
| index([start, end]) | Trả về chỉ số đầu tiên của phần tử x trong tuple (từ start đến end) | (1,2,3).index(2) # 1 |

Ví dụ

t = (10, 20, 30, 20, 40)

print(t.count(20)) # Output: 2

print(t.index(30)) # Output: 2

print(t.index(20, 2)) # Output: 3 (tìm từ vị trí index 2 trở đi)

### Dictionary (Từ điển)

* Lưu trữ dữ liệu theo cặp key-value (khóa-giá trị)
* Sử dụng dấu ngoặc nhọn {}
* Truy cập nhanh thông qua key

| **Phương thức** | **Mô tả** | **Ví dụ sử dụng** |
| --- | --- | --- |
| dict.clear() | Xóa tất cả các cặp key-value trong dict | d.clear() |
| dict.copy() | Trả về bản sao shallow của dict | d2 = d.copy() |
| dict.get(key, [default]) | Lấy giá trị theo key, nếu key không tồn tại trả về default | value = d.get('key', 'default') |
| dict.items() | Trả về view các cặp (key, value) | for k, v in d.items(): print(k, v) |
| dict.keys() | Trả về view các key | keys = d.keys() |
| dict.pop(key,[default]) | Lấy và xóa phần tử theo key, nếu không có trả về default hoặc lỗi | d.pop('key', None) |
| dict.popitem() | Lấy và xóa phần tử cuối cùng (trong Python 3.7+) | key, val = d.popitem() |
| dict.setdefault(key, [default]) | Lấy giá trị theo key hoặc thêm key với default nếu chưa có | d.setdefault('key', 0) |
| dict.update([other]) | Cập nhật dict bằng cặp key-value từ dict hoặc iterable | d.update({'newkey': 10}) |
| dict.values() | Trả về view các giá trị | values = d.values() |

Ví dụ:

d = {'a': 1, 'b': 2, 'c': 3}

print(d.get('b')) # 2

print(d.get('x', 0)) # 0

d.setdefault('d', 4)

print(d) # {'a':1, 'b':2, 'c':3, 'd':4}

for k, v in d.items():

print(k, v)

d.pop('a')

print(d) # {'b':2, 'c':3, 'd':4}

d.update({'e': 5, 'f': 6})

print(d) # {'b':2, 'c':3, 'd':4, 'e':5, 'f':6}

### Set (Tập hợp)

* Tập hợp các phần tử duy nhất, không có thứ tự
* Tự động loại bỏ phần tử trùng lặp
* Hỗ trợ các phép toán tập hợp như hợp, giao, hiệu

| **Phương thức** | **Mô tả** | **Ví dụ sử dụng** |
| --- | --- | --- |
| add(elem) | Thêm phần tử elem vào set nếu chưa có | s.add(5) |
| remove(elem) | Xóa phần tử elem khỏi set, lỗi nếu không tồn tại | s.remove(3) |
| discard(elem) | Xóa phần tử elem khỏi set, không lỗi nếu không tồn tại | s.discard(3) |
| pop() | Lấy và xóa phần tử ngẫu nhiên khỏi set | x = s.pop() |
| clear() | Xóa hết các phần tử trong set | s.clear() |
| copy() | Trả về bản sao của set | s2 = s.copy() |
| union(\*others) | Trả về tập hợp hợp của set và sets khác | s.union(t1, t2) |
| update(\*others) | Thêm các phần tử từ các sets hoặc iterable khác vào set | s.update(t1, t2) |
| intersection(\*others) | Trả về phần giao của set với các sets khác | s.intersection(t1, t2) |
| intersection\_update(\*others) | Giữ lại phần giao trong chính set, xóa các phần không thuộc giao | s.intersection\_update(t1) |
| difference(\*others) | Trả về phần hiệu (phần tử chỉ có trong set mà không trong others) | s.difference(t1) |
| difference\_update(\*others) | Giữ lại phần hiệu trong chính set | s.difference\_update(t1) |
| symmetric\_difference(other) | Trả về phần khác nhau giữa 2 sets | s.symmetric\_difference(t) |
| symmetric\_difference\_update(other) | Cập nhật set thành phần khác nhau với other | s.symmetric\_difference\_update(t) |
| issubset(other) | Kiểm tra xem set có phải con tập của other | s.issubset(t) |
| issuperset(other) | Kiểm tra xem set có phủ rộng other | s.issuperset(t) |
| isdisjoint(other) | Kiểm tra xem set có phần tử chung nào với other không | s.isdisjoint(t) |

s = {1, 2, 3}

s.add(4) # s = {1, 2, 3, 4}

s.remove(2) # s = {1, 3, 4}

s.discard(5) # không lỗi

x = s.pop() # lấy phần tử ngẫu nhiên và xóa

print(s.union({3,5})) # {1,3,4,5}

print(s.intersection({1,4,5})) # {1,4}

print(s.isdisjoint({6,7})) # True

## Cấu trúc điều khiển trong Python

### Câu lệnh điều kiện (if/elif/else)

* Cho phép chương trình đưa ra quyết định dựa trên điều kiện
* Sử dụng indentation (thụt lề) để xác định khối lệnh

### Vòng lặp

* **for loop**: Lặp qua các phần tử trong một iterable (list, string, range...)
* **while loop**: Lặp khi điều kiện còn đúng

**Ví dụ**:

**For loop**:

A screenshot of a diagram

AI-generated content may be incorrect.

**While loop:**

A screenshot of a computer code

AI-generated content may be incorrect.

## Hàm (Functions) trong Python

### Tại sao cần sử dụng hàm?

* Tái sử dụng code (DRY - Don't Repeat Yourself)
* Tổ chức code rõ ràng, dễ bảo trì
* Chia nhỏ vấn đề phức tạp thành các phần nhỏ

### Thành phần của hàm:

* Tên hàm: đặt tên có ý nghĩa
* Tham số (parameters): đầu vào của hàm
* Giá trị trả về (return): kết quả của hàm

## Class và object

*Xem thêm tại:* [*https://www.w3schools.com/python/python\_classes.asp*](https://www.w3schools.com/python/python_classes.asp)

### Class

Trong Python, Class là một khuôn mẫu (bản thiết kế) định nghĩa các thuộc tính (dữ liệu) và phương thức (hành vi) cho một loại đối tượng nhất định. Ngược lại, Object (hoặc đối tượng) là một thể hiện cụ thể, một bản sao của Class đó, chứa dữ liệu riêng biệt và có thể thực hiện các hành vi được định nghĩa trong Class. Hiểu rõ sự khác biệt này là bước quan trọng để nắm vững lập trình hướng đối tượng (OOP).

* **Bản thiết kế:**

Class giống như một bản phác thảo chi tiết về một "thứ" nào đó, mô tả nó sẽ có những đặc điểm gì (thuộc tính) và có thể làm gì (phương thức).

* **Định nghĩa:**

Nó cung cấp cấu trúc dữ liệu và hành vi chung cho tất cả các đối tượng được tạo ra từ Class đó.

### Object (Đối tượng)

Object là một thực thể thực tế được tạo ra từ Class.

* **Dữ liệu riêng:**

Mỗi Object có các giá trị riêng cho các thuộc tính được định nghĩa trong Class.

* **Thực thi hành vi:**

Object sử dụng các phương thức được định nghĩa trong Class để thực hiện các hành động.

* **Ví dụ:**

Dựa trên Class XeHoi, bạn có thể tạo ra các Object như xe\_cua\_toi = XeHoi(mau\_sac='do', hang\_xe='Toyota') hoặc xe\_cua\_ban = XeHoi(mau\_sac='xanh', hang\_xe='Honda'). Mỗi Object này có màu sắc và hãng xe riêng.

Ví dụ minh họa trong Python

# Định nghĩa Class XeHoi

class XeHoi:

# Phương thức khởi tạo (hàm tạo)

def \_\_init\_\_(self, hang, mau\_sac):

self.hang = hang # Thuộc tính instance

self.mau\_sac = mau\_sac # Thuộc tính instance

# Phương thức của Class

def bat\_dau\_chay(self):  
 print(f"Xe {self.hang} màu {self.mau\_sac} đang chạy...")  
  
# Tạo các Object (thể hiện của Class XeHoi)

xe\_toyota = XeHoi("Toyota", "Đỏ")

xe\_honda = XeHoi("Honda", "Xanh")  
  
# Sử dụng các thuộc tính và phương thức của Object

print(f"Chiếc xe này là {xe\_toyota.hang} màu {xe\_toyota.mau\_sac}")

xe\_toyota.bat\_dau\_chay()  
  
print(f"Chiếc xe kia là {xe\_honda.hang} màu {xe\_honda.mau\_sac}")  
xe\_honda.bat\_dau\_chay()

Trong ví dụ này:

* XeHoi là Class (khuôn mẫu).
* xe\_toyota và xe\_honda là các Object (thể hiện cụ thể) của Class XeHoi.
* hang và mau\_sac là các thuộc tính (thông tin) của mỗi Object.
* \_\_init\_\_ và bat\_dau\_chay() là các phương thức (hành vi) mà các Object có thể sử dụng

### Tham số self

Tham số self là một tham chiếu tới phiên bản hiện tại của class. Nó thường được dùng để truy cập các biến thuộc về class đó.

Không nhất thiết phải đặt tên nó là self. Bạn có thể gọi nó bất cứ khi nào muốn nhưng nó phải là tham số đầu tiên của hàm bất kỳ trong class.

* Ví dụ: Dùng từ bduoss và abc thay cho self:

class Person:

def \_\_init\_\_(bduoss, name, age):

bduoss.name = name

bduoss.age = age

def myfunc(abc):

print("Xin chào, tên tôi là " + abc.name)

p1 = Person("Quantrimang.com", 16)

p1.myfunc()

**Quy ước về các biến Public/Private/Protected**

| **Mức độ** | **Quy ước đặt tên** | **Tính chất** | **Có name mangling?** | **Có ngăn truy cập?** |
| --- | --- | --- | --- | --- |
| Public | Không có dấu gạch dưới | Có thể truy cập từ mọi nơi | Không | Không |
| Protected | Một dấu gạch dưới \_ | Mong muốn truy cập trong class hoặc subclass | Không | Không (quy ước) |
| Private | Hai dấu gạch dưới \_\_ | Khó truy cập từ bên ngoài class, name mangling | Có | Gần như Có |

## Xử lý ngoại lệ (Exception Handling)

**Tại sao cần xử lý lỗi?**

* Chương trình thực tế luôn có khả năng gặp lỗi
* Xử lý lỗi giúp chương trình không bị crash
* Cung cấp thông báo lỗi thân thiện cho người dùng

**Cấu trúc try-except-finally:**

* try: Khối code có thể gây lỗi
* except: Xử lý khi có lỗi xảy ra
* finally: Luôn được thực thi (dùng để dọn dẹp tài nguyên)

# Hướng dẫn thực hành

## Python basic

**Tạo tài khoản trên trang, và đăng kí khóa học miễn phí:**

<https://codelearn.io/learning/python-co-ban>

## Khởi động và chuẩn bị môi trường

**Bước 1: Kết nối Ubuntu qua SSH trên VSCode (Xem file Thực hành buổi 1)**

**Bước 2: Mở Terminal, Kích hoạt môi trường ảo**

cd workspace/todo\_project

source venv/bin/activate

**Bước 3: Tạo file thực hành**

Tạo 6 file để luyện tập thực hành:

* list\_practice.py: Thao tác với danh sách
* dict\_practice.py: Quản lý thông tin bằng dictionary
* loops\_conditions.py: Vòng lặp và điều kiện
* functions\_practice.py: Viết và sử dụng hàm
* exception\_practice.py: Xử lý lỗi an toàn
* todo\_manager.py: **Dự án mini tích hợp** - Quản lý công việc

## Thực hành với List (Danh sách)

Thực hành trên file **list\_practice.py**với nội dung:

# Khởi tạo list

fruits = ["apple", "banana", "orange"]

numbers = [1, 2, 3, 4, 5]

mixed\_list = ["hello", 42, True, 3.14]

# In ra list

print("Fruits:", fruits)

print("Numbers:", numbers)

# Thêm phần tử

fruits.append("grape")

print("After append:", fruits)

# Truy cập phần tử

print("First fruit:", fruits[0])

print("Last fruit:", fruits[-1])

# Thay đổi phần tử

fruits[1] = "mango"

print("After change:", fruits)

# Lấy độ dài list

print("Number of fruits:", len(fruits))

# Kiểm tra phần tử có trong list

if "apple" in fruits:

print("Apple is in the list")

# Xóa phần tử

fruits.remove("apple")

print("After remove:", fruits)

Chạy và kiểm tra kết quả:

python3 list\_practice.py

## Thực hành với Dictionary (Từ điển)

**Tạo file dict\_practice.py:**

# Khởi tạo dictionary - thông tin sinh viên

student = {

"name": "Nguyen Van A",

"age": 20,

"major": "Computer Science",

"gpa": 3.5

}

# In ra thông tin

print("Student info:", student)

# Truy cập giá trị

print("Name:", student["name"])

print("Age:", student["age"])

# Thêm thông tin mới

student["phone"] = "0123456789"

print("After adding phone:", student)

# Sửa thông tin

student["gpa"] = 3.7

print("After updating GPA:", student)

# Lấy tất cả keys và values

print("Keys:", list(student.keys()))

print("Values:", list(student.values()))

# Kiểm tra key có tồn tại

if "email" in student:

print("Email exists")

else:

print("Email not found")

# Xóa thông tin

del student["phone"]

print("After deleting phone:", student)

## Thực hành vòng lặp và điều kiện

**Tạo file loops\_conditions.py:**

# Vòng lặp for với list

fruits = ["apple", "banana", "orange"]

print("=== For loop with list ===")

for fruit in fruits:

print(f"I like {fruit}")

# Vòng lặp for với range

print("\n=== For loop with range ===")

for i in range(5):

print(f"Number: {i}")

# Vòng lặp while

print("\n=== While loop ===")

count = 0

while count < 3:

print(f"Count: {count}")

count += 1

# Câu lệnh điều kiện

print("\n=== Conditional statements ===")

score = 85

if score >= 90:

grade = "A"

elif score >= 80:

grade = "B"

elif score >= 70:

grade = "C"

elif score >= 60:

grade = "D"

else:

grade = "F"

print(f"Score: {score}, Grade: {grade}")

# Kết hợp điều kiện và vòng lặp

print("\n=== Numbers classification ===")

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

for num in numbers:

if num % 2 == 0:

print(f"{num} is even")

else:

print(f"{num} is odd")

## Thực hành với Functions (Hàm)

**Tạo file functions\_practice.py:**

# Hàm đơn giản không có tham số

def greet():

print("Hello, World!")

# Hàm có tham số

def greet\_person(name):

print(f"Hello, {name}!")

# Hàm có giá trị trả về

def add\_numbers(a, b):

result = a + b

return result

# Hàm tính điểm trung bình

def calculate\_average(scores):

if len(scores) == 0:

return 0

total = sum(scores)

average = total / len(scores)

return average

# Hàm kiểm tra số chẵn lẻ

def check\_even\_odd(number):

if number % 2 == 0:

return "even"

else:

return "odd"

# Hàm quản lý thông tin sinh viên

def create\_student(name, age, major):

student = {

"name": name,

"age": age,

"major": major,

"courses": []

}

return student

def add\_course(student, course):

student["courses"].append(course)

print(f"Added {course} to {student['name']}'s courses")

# Test các hàm

print("=== Testing functions ===")

greet()

greet\_person("Alice")

result = add\_numbers(5, 3)

print(f"5 + 3 = {result}")

scores = [85, 90, 78, 92, 88]

avg = calculate\_average(scores)

print(f"Average score: {avg:.2f}")

print(f"7 is {check\_even\_odd(7)}")

print(f"8 is {check\_even\_odd(8)}")

# Tạo và quản lý thông tin sinh viên

student1 = create\_student("John Doe", 21, "Computer Science")

print("Student created:", student1)

add\_course(student1, "Python Programming")

add\_course(student1, "Data Structures")

print("Final student info:", student1)

## Thực hành Exception Handling (Xử lý lỗi)

**Tạo file exception\_practice.py:**

# Xử lý lỗi chia cho 0

def safe\_divide(a, b):

try:

result = a / b

return result

except ZeroDivisionError:

print("Error: Cannot divide by zero!")

return None

except TypeError:

print("Error: Please provide numbers only!")

return None

# Xử lý lỗi truy cập list

def safe\_get\_item(my\_list, index):

try:

item = my\_list[index]

return item

except IndexError:

print(f"Error: Index {index} is out of range!")

return None

except TypeError:

print("Error: Index must be a number!")

return None

# Xử lý lỗi chuyển đổi kiểu dữ liệu

def safe\_convert\_to\_int(value):

try:

result = int(value)

return result

except ValueError:

print(f"Error: '{value}' cannot be converted to integer!")

return None

# Xử lý lỗi khi đọc file

def safe\_read\_file(filename):

try:

with open(filename, 'r') as file:

content = file.read()

return content

except FileNotFoundError:

print(f"Error: File '{filename}' not found!")

return None

except PermissionError:

print(f"Error: No permission to read '{filename}'!")

return None

finally:

print("File operation completed (success or error)")

# Test exception handling

print("=== Testing Exception Handling ===")

# Test safe divide

print("10 / 2 =", safe\_divide(10, 2))

print("10 / 0 =", safe\_divide(10, 0))

print("10 / 'a' =", safe\_divide(10, 'a'))

# Test safe get item

my\_list = [1, 2, 3, 4, 5]

print("Item at index 2:", safe\_get\_item(my\_list, 2))

print("Item at index 10:", safe\_get\_item(my\_list, 10))

# Test safe convert

print("Convert '123':", safe\_convert\_to\_int('123'))

print("Convert 'abc':", safe\_convert\_to\_int('abc'))

# Test safe read file

content = safe\_read\_file('nonexistent.txt')

**BONUS**

**Pythonic Code là gì?**

Pythonic nghĩa là tận dụng tối đa tính năng và đặc điểm riêng của Python để viết code. Code Pythonic dễ đọc, dễ hiểu, ngắn gọn như đọc tiếng Anh, đồng thời tuân thủ các quy ước và triết lý của Python.

A screenshot of a computer

AI-generated content may be incorrect.

**Indexes và Slices**

Python cung cấp cách truy cập mạnh mẽ vào các phần tử trong sequences (list, tuple, string) thông qua indexes và slicing. Cú pháp slicing sequence[start:stop:step] giúp thao tác với dữ liệu linh hoạt và Pythonic.

A screenshot of a computer

AI-generated content may be incorrect.

## Bài tập tích hợp: Quản lý danh sách công việc đơn giản

File**todo\_manager.py:**

# Danh sách lưu trữ các công việc

tasks = []

def add\_task(task\_name):

"""Thêm công việc mới"""

task = {

"id": len(tasks) + 1,

"name": task\_name,

"completed": False

}

tasks.append(task)

print(f"✅ Added task: {task\_name}")

def list\_tasks():

"""Hiển thị tất cả công việc"""

if not tasks:

print("📝 No tasks available")

return

print("📋 Task List:")

for task in tasks:

status = "✅" if task["completed"] else "⏳"

print(f"{task['id']}. {task['name']} {status}")

def complete\_task(task\_id):

"""Đánh dấu công việc hoàn thành"""

for task in tasks:

if task["id"] == task\_id:

task["completed"] = True

print(f"🎉 Completed task: {task['name']}")

return

print(f"❌ Task with ID {task\_id} not found")

def delete\_task(task\_id):

"""Xóa công việc"""

for i, task in enumerate(tasks):

if task["id"] == task\_id:

removed\_task = tasks.pop(i)

print(f"🗑️ Deleted task: {removed\_task['name']}")

return

print(f"❌ Task with ID {task\_id} not found")

def get\_task\_stats():

"""Thống kê công việc"""

total = len(tasks)

completed = sum(1 for task in tasks if task["completed"])

pending = total - completed

print(f"📊 Task Statistics:")

print(f" Total: {total}")

print(f" Completed: {completed}")

print(f" Pending: {pending}")

# Demo chương trình

def main():

print("🚀 Welcome to Simple TODO Manager!")

# Thêm một số task mẫu

add\_task("Learn Python basics")

add\_task("Practice with lists and dictionaries")

add\_task("Write first function")

add\_task("Handle exceptions properly")

# Hiển thị danh sách

list\_tasks()

# Hoàn thành một task

complete\_task(1)

complete\_task(3)

# Hiển thị lại danh sách

print("\n" + "="\*30)

list\_tasks()

# Thống kê

print("\n" + "="\*30)

get\_task\_stats()

# Xóa một task

print("\n" + "="\*30)

delete\_task(2)

# Hiển thị cuối cùng

print("\n" + "="\*30)

list\_tasks()

get\_task\_stats()

# Chạy chương trình

if \_\_name\_\_ == "\_\_main\_\_":

main()

## Object-Oriented Programming basic

from enum import Enum  
from datetime import datetime  
from typing import List, Optional  
  
class TaskStatus(Enum):  
 TODO = "todo"  
 IN\_PROGRESS = "in\_progress"  
 DONE = "done"  
  
class Priority(Enum):  
 LOW = "low"  
 MEDIUM = "medium"  
 HIGH = "high"  
  
class UserRole(Enum):  
 ADMIN = "admin"  
 USER = "user"  
  
class BaseEntity:  
 """Base class cho tất cả entities"""  
 def \_\_init\_\_(self):  
 self.id = self.\_generate\_id()  
 self.created\_at = datetime.now()  
 self.updated\_at = datetime.now()  
   
 def \_generate\_id(self):  
 import uuid  
 return str(uuid.uuid4())[:8]  
   
 def update\_timestamp(self):  
 self.updated\_at = datetime.now()

**9.3 User Class**

class User(BaseEntity):  
 def \_\_init\_\_(self, username: str, email: str, full\_name: str = None):  
 super().\_\_init\_\_()  
 self.username = username  
 self.email = email  
 self.full\_name = full\_name or username  
 self.role = UserRole.USER  
 self.is\_active = True  
 self.\_password\_hash = None  
   
 def set\_password(self, password: str):  
 """Set password (sẽ hash trong thực tế)"""  
 # Trong thực tế sẽ dùng bcrypt  
 self.\_password\_hash = hash(password)  
 self.update\_timestamp()  
   
 def verify\_password(self, password: str) -> bool:  
 """Verify password"""  
 return self.\_password\_hash == hash(password)  
   
 def promote\_to\_admin(self):  
 """Promote user to admin"""  
 self.role = UserRole.ADMIN  
 self.update\_timestamp()  
   
 def deactivate(self):  
 """Deactivate user"""  
 self.is\_active = False  
 self.update\_timestamp()  
   
 def \_\_str\_\_(self):  
 return f"User({self.username}, {self.role.value})"  
   
 def \_\_repr\_\_(self):  
 return f"User(id={self.id}, username={self.username}, role={self.role.value})"

**9.4 Board Class**

class Board(BaseEntity):  
 def \_\_init\_\_(self, name: str, description: str = None, owner: User = None):  
 super().\_\_init\_\_()  
 self.name = name  
 self.description = description or ""  
 self.owner = owner  
 self.is\_public = False  
 self.tasks: List['Task'] = []  
   
 def add\_task(self, task: 'Task'):  
 """Add task to board"""  
 if task not in self.tasks:  
 self.tasks.append(task)  
 task.board = self  
 self.update\_timestamp()  
   
 def remove\_task(self, task: 'Task'):  
 """Remove task from board"""  
 if task in self.tasks:  
 self.tasks.remove(task)  
 task.board = None  
 self.update\_timestamp()  
   
 def get\_tasks\_by\_status(self, status: TaskStatus) -> List['Task']:  
 """Get all tasks with specific status"""  
 return [task for task in self.tasks if task.status == status]  
   
 def get\_tasks\_by\_priority(self, priority: Priority) -> List['Task']:  
 """Get all tasks with specific priority"""  
 return [task for task in self.tasks if task.priority == priority]  
   
 def get\_task\_count(self) -> dict:  
 """Get count of tasks by status"""  
 return {  
 "todo": len(self.get\_tasks\_by\_status(TaskStatus.TODO)),  
 "in\_progress": len(self.get\_tasks\_by\_status(TaskStatus.IN\_PROGRESS)),  
 "done": len(self.get\_tasks\_by\_status(TaskStatus.DONE)),  
 "total": len(self.tasks)  
 }  
   
 def set\_public(self, is\_public: bool):  
 """Set board visibility"""  
 self.is\_public = is\_public  
 self.update\_timestamp()  
   
 def \_\_str\_\_(self):  
 return f"Board({self.name}, {len(self.tasks)} tasks)"

**9.5 Task Class**

class Task(BaseEntity):  
 def \_\_init\_\_(self, title: str, description: str = None):  
 super().\_\_init\_\_()  
 self.title = title  
 self.description = description or ""  
 self.status = TaskStatus.TODO  
 self.priority = Priority.MEDIUM  
 self.assigned\_to: Optional[User] = None  
 self.board: Optional[Board] = None  
 self.position = 0  
 self.due\_date: Optional[datetime] = None  
   
 def assign\_to(self, user: User):  
 """Assign task to user"""  
 self.assigned\_to = user  
 self.update\_timestamp()  
   
 def unassign(self):  
 """Unassign task"""  
 self.assigned\_to = None  
 self.update\_timestamp()  
   
 def set\_priority(self, priority: Priority):  
 """Set task priority"""  
 self.priority = priority  
 self.update\_timestamp()  
   
 def move\_to\_status(self, new\_status: TaskStatus):  
 """Move task to different status"""  
 old\_status = self.status  
 self.status = new\_status  
 self.update\_timestamp()  
 print(f"Task '{self.title}' moved from {old\_status.value} to {new\_status.value}")  
   
 def set\_due\_date(self, due\_date: datetime):  
 """Set due date for task"""  
 self.due\_date = due\_date  
 self.update\_timestamp()  
   
 def is\_overdue(self) -> bool:  
 """Check if task is overdue"""  
 if self.due\_date and self.status != TaskStatus.DONE:  
 return datetime.now() > self.due\_date  
 return False  
   
 def update\_content(self, title: str = None, description: str = None):  
 """Update task content"""  
 if title:  
 self.title = title  
 if description:  
 self.description = description  
 self.update\_timestamp()  
   
 def \_\_str\_\_(self):  
 assigned = f" -> {self.assigned\_to.username}" if self.assigned\_to else ""  
 return f"Task({self.title}, {self.status.value}, {self.priority.value}{assigned})"

**10. Kanban Manager - Main Application Class**

class KanbanManager:  
 def \_\_init\_\_(self):  
 self.users: List[User] = []  
 self.boards: List[Board] = []  
 self.current\_user: Optional[User] = None  
   
 # User Management  
 def register\_user(self, username: str, email: str, password: str, full\_name: str = None) -> User:  
 """Register new user"""  
 # Check if username exists  
 if any(user.username == username for user in self.users):  
 raise ValueError(f"Username '{username}' already exists")  
   
 user = User(username, email, full\_name)  
 user.set\_password(password)  
 self.users.append(user)  
 return user  
   
 def login(self, username: str, password: str) -> bool:  
 """Login user"""  
 user = self.get\_user\_by\_username(username)  
 if user and user.verify\_password(password) and user.is\_active:  
 self.current\_user = user  
 return True  
 return False  
   
 def logout(self):  
 """Logout current user"""  
 self.current\_user = None  
   
 def get\_user\_by\_username(self, username: str) -> Optional[User]:  
 """Get user by username"""  
 return next((user for user in self.users if user.username == username), None)  
   
 # Board Management  
 def create\_board(self, name: str, description: str = None) -> Board:  
 """Create new board"""  
 if not self.current\_user:  
 raise ValueError("Must be logged in to create board")  
   
 board = Board(name, description, self.current\_user)  
 self.boards.append(board)  
 return board  
   
 def get\_user\_boards(self, user: User = None) -> List[Board]:  
 """Get boards owned by user"""  
 user = user or self.current\_user  
 if not user:  
 return []  
 return [board for board in self.boards if board.owner == user]  
   
 def get\_public\_boards(self) -> List[Board]:  
 """Get all public boards"""  
 return [board for board in self.boards if board.is\_public]  
   
 # Task Management  
 def create\_task(self, board: Board, title: str, description: str = None) -> Task:  
 """Create task in board"""  
 if not self.current\_user:  
 raise ValueError("Must be logged in to create task")  
   
 # Check if user has permission (owner or admin)  
 if board.owner != self.current\_user and self.current\_user.role != UserRole.ADMIN:  
 if not board.is\_public:  
 raise ValueError("No permission to create task in this board")  
   
 task = Task(title, description)  
 board.add\_task(task)  
 return task  
   
 def move\_task(self, task: Task, new\_status: TaskStatus):  
 """Move task to different status"""  
 if not self.current\_user:  
 raise ValueError("Must be logged in to move task")  
   
 task.move\_to\_status(new\_status)  
   
 def assign\_task(self, task: Task, user: User):  
 """Assign task to user"""  
 if not self.current\_user:  
 raise ValueError("Must be logged in to assign task")  
   
 task.assign\_to(user)  
   
 # Utility methods  
 def get\_current\_user\_info(self) -> str:  
 """Get current user info"""  
 if not self.current\_user:  
 return "No user logged in"  
   
 user\_boards = len(self.get\_user\_boards())  
 total\_tasks = sum(len(board.tasks) for board in self.get\_user\_boards())  
   
 return f"""  
Current User: {self.current\_user.full\_name} (@{self.current\_user.username})  
Role: {self.current\_user.role.value}  
Boards: {user\_boards}  
Total Tasks: {total\_tasks}  
 """.strip()  
   
 def get\_system\_stats(self) -> str:  
 """Get system statistics"""  
 active\_users = len([u for u in self.users if u.is\_active])  
 total\_boards = len(self.boards)  
 public\_boards = len(self.get\_public\_boards())  
 total\_tasks = sum(len(board.tasks) for board in self.boards)  
   
 return f"""  
System Statistics:  
- Users: {active\_users} active, {len(self.users)} total  
- Boards: {total\_boards} total, {public\_boards} public  
- Tasks: {total\_tasks} total  
 """.strip()

# Bài tập

**Bài 1:** Viết chương trình quản lý thông tin sinh viên với các chức năng:

* Thêm sinh viên mới (tên, tuổi, điểm trung bình)
* Hiển thị danh sách sinh viên
* Tìm sinh viên theo tên
* Xóa sinh viên
* Tính điểm trung bình của lớp

**Bài 2:** Viết chương trình quản lý từ điển Anh-Việt đơn giản:

* Thêm từ mới
* Tra cứu từ
* Hiển thị tất cả từ
* Xóa từ

**Bài 3:** Sử dụng exception handling để viết chương trình an toàn:

* Nhập số từ người dùng
* Kiểm tra số chẵn/lẻ
* Xử lý trường hợp người dùng nhập không phải số