Python Programming - Lists (Danh Sách)

> **Mục tiêu học tập**: Làm chủ Lists - cấu trúc dữ liệu mạnh mẽ và linh hoạt nhất trong Python, từ thao tác cơ bản đến list comprehension và nested lists.

1. Giới Thiệu về Lists

1.1. List là gì?

🤔 Đặt vấn đề

Giả sử bạn cần lưu điểm của 50 học sinh. Dùng 50 biến riêng biệt?

```
# ❌ Cách không hiệu quả
score1 = 8.5
score2 = 7.0
score3 = 9.5
# ... 47 biến nữa???
```

Làm sao để quản lý nhiều giá trị liên quan một cách hiệu quả?

💡 Giải quyết: Lists

List là một **cấu trúc dữ liệu có thứ tự** (ordered collection) để lưu trữ nhiều giá trị.

```
# ☑ Cách hiệu quả
scores = [8.5, 7.0, 9.5, 6.0, 8.0]
```

Đặc điểm của List:

- ☑ **Ordered** (có thứ tự): Giữ nguyên thứ tự phần tử
- ☑ **Mutable** (có thể thay đổi): Có thể sửa, thêm, xóa phần tử
- ☑ **Heterogeneous** (không đồng nhất): Có thể chứa nhiều kiểu dữ liệu khác nhau
- ☑ **Dynamic** (động): Kích thước có thể thay đổi
- ☑ **Allow duplicates** (cho phép trùng lặp)

1.2. So Sánh List vs String

Tiêu chí	List	String
Mutable	☑ Có thể thay đổi	❌ Không thể thay đổi
Elements	Bất kỳ kiểu nào	Chỉ ký tự
Syntax	`[1, 2, 3]`	`"abc"`

Tiêu chí	List	String
Use case	Lưu collection dữ liệu	Lưu text

```
# String - immutable
text = "Python"
text[0] = "J"  # ❌ TypeError

# List - mutable
numbers = [1, 2, 3]
numbers[0] = 10  # ☑ OK
print(numbers)  # [10, 2, 3]
```

---

2. Tạo Lists

2.1. List Literals

```
# List rỗng
empty_list = []

# List số nguyên
numbers = [1, 2, 3, 4, 5]

# List chuỗi
names = ["An", "Bình", "Chi"]

# List hỗn hợp (nhiều kiểu dữ liệu)
mixed = [1, "Python", 3.14, True, None]

# List chứa list (nested)
matrix = [[1, 2], [3, 4], [5, 6]]
```

2.2. Hàm `list()`

Chuyển đổi iterable thành list.

```
# Từ string
chars = list("Python")
print(chars)  # ['P', 'y', 't', 'h', 'o', 'n']

# Từ range
numbers = list(range(5))
print(numbers)  # [0, 1, 2, 3, 4]

numbers = list(range(1, 11, 2))
print(numbers)  # [1, 3, 5, 7, 9]
```

```
# Từ tuple
coordinates = (10, 20, 30)
points = list(coordinates)
print(points)  # [10, 20, 30]
```

2.3. List Comprehension (Sẽ nói chi tiết sau)

```
# Tạo list bình phương
squares = [x**2 for x in range(5)]
print(squares)  # [0, 1, 4, 9, 16]
```

---

3. Truy Cập Elements

3.1. Indexing (Giống String)

```
fruits = ["apple", "banana", "orange", "mango"]
#         0        1         2         3
#        -4       -3        -2        -1

# Positive indexing
print(fruits[0])   # apple
print(fruits[2])   # orange

# Negative indexing
print(fruits[-1])  # mango (phần tử cuối)
print(fruits[-2])  # orange

# Truy cập phần tử cuối
last_item = fruits[len(fruits) - 1]  # Cách 1
last_item = fruits[-1]               # Cách 2 (ngắn gọn hơn)
```

3.2. Slicing (Giống String)

```
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# Lấy 3 phần tử đầu
print(numbers[:3])      # [0, 1, 2]

# Lấy từ index 3 đến cuối
print(numbers[3:])      # [3, 4, 5, 6, 7, 8, 9]

# Lấy từ index 2 đến 7
print(numbers[2:8])     # [2, 3, 4, 5, 6, 7]
```

```
# Lấy mỗi phần tử thứ 2
print(numbers[::2])     # [0, 2, 4, 6, 8]

# Đảo ngược list
print(numbers[::-1])    # [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

# Lấy 5 phần tử cuối
print(numbers[-5:])     # [5, 6, 7, 8, 9]
```

Ví dụ thực tế: Xử lý dữ liệu

```
# Lấy 10 kết quả đầu tiên
search_results = ["result1", "result2", "result3", ...]
top_10 = search_results[:10]

# Phân trang (pagination)
page = 2
page_size = 20
start = (page - 1) * page_size
end = start + page_size
items_on_page = all_items[start:end]
```

---

4. Thay Đổi Elements (Mutable)

4.1. Thay Đổi Một Phần Tử

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

# Thay đổi phần tử thứ 2
fruits[1] = "grape"
print(fruits)  # ['apple', 'grape', 'orange']

# Thay đổi phần tử cuối
fruits[-1] = "mango"
print(fruits)  # ['apple', 'grape', 'mango']
```

4.2. Thay Đổi Nhiều Phần Tử (Slicing)

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

# Thay đổi 3 phần tử đầu
numbers[:3] = [10, 20, 30]
print(numbers)  # [10, 20, 30, 4, 5]

# Thay đổi bằng list khác độ dài
numbers[1:3] = [100, 200, 300, 400]
```

```
print(numbers)  # [10, 100, 200, 300, 400, 4, 5]

# Xóa phần tử bằng cách gán []
numbers[2:4] = []
print(numbers)  # [10, 100, 4, 5]
```

---

5. List Methods (Phương Thức)

5.1. Thêm Phần Tử

`.append()` - Thêm vào cuối

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

fruits.append("orange")
print(fruits)  # ['apple', 'banana', 'orange']

fruits.append("mango")
print(fruits)  # ['apple', 'banana', 'orange', 'mango']

# Chỉ thêm được 1 phần tử
fruits.append(["grape", "kiwi"])
print(fruits)  # ['apple', 'banana', 'orange', 'mango', ['grape', 'kiwi']]
#                                                    ↑ list lồng nhau
```

`.extend()` - Thêm nhiều phần tử

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

# Thêm nhiều phần tử từ iterable
fruits.extend(["orange", "mango"])
print(fruits)  # ['apple', 'banana', 'orange', 'mango']

# Có thể extend từ string (mỗi ký tự là 1 phần tử)
letters = ['a', 'b']
letters.extend("cd")
print(letters)  # ['a', 'b', 'c', 'd']
```

So sánh `append()` vs `extend()`:

```
list1 = [1, 2, 3]
list1.append([4, 5])
print(list1)  # [1, 2, 3, [4, 5]]

list2 = [1, 2, 3]
```

```
list2.extend([4, 5])
print(list2)  # [1, 2, 3, 4, 5]
```

`.insert()` - Thêm vào vị trí cụ thể

```
fruits = ["apple", "orange", "mango"]

# Chèn "banana" vào index 1
fruits.insert(1, "banana")
print(fruits)  # ['apple', 'banana', 'orange', 'mango']

# Chèn vào đầu
fruits.insert(0, "grape")
print(fruits)  # ['grape', 'apple', 'banana', 'orange', 'mango']

# Chèn vào cuối (giống append)
fruits.insert(len(fruits), "kiwi")
print(fruits)  # ['grape', 'apple', 'banana', 'orange', 'mango', 'kiwi']
```

5.2. Xóa Phần Tử

`.remove()` - Xóa theo giá trị

Xóa **lần xuất hiện đầu tiên** của giá trị.

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

fruits.remove("banana")
print(fruits)  # ['apple', 'orange', 'banana']
#                       ↑ chỉ xóa cái đầu tiên

# Nếu không tồn tại → ValueError
try:
    fruits.remove("grape")
except ValueError:
    print("Không tìm thấy!")
```

`.pop()` - Xóa theo index và trả về giá trị

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

# Pop phần tử cuối (mặc định)
last = fruits.pop()
print(last)    # orange
print(fruits)  # ['apple', 'banana']
```

```
# Pop phần tử tại index cụ thể
first = fruits.pop(0)
print(first)   # apple
print(fruits)  # ['banana']

# Ứng dụng: Stack (LIFO)
stack = []
stack.append(1)  # Push
stack.append(2)
stack.append(3)
print(stack.pop())  # 3 - Pop
print(stack.pop())  # 2
```

`.clear()` - Xóa tất cả

```
fruits = ["apple", "banana", "orange"]
fruits.clear()
print(fruits)  # []
```

`del` statement

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

# Xóa 1 phần tử
del fruits[1]
print(fruits)  # ['apple', 'orange', 'mango']

# Xóa slice
del fruits[:2]
print(fruits)  # ['mango']

# Xóa toàn bộ list
del fruits
# print(fruits)  # NameError: name 'fruits' is not defined
```

5.3. Tìm Kiếm và Đếm

`.index()` - Tìm vị trí

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

# Tìm vị trí đầu tiên
pos = fruits.index("banana")
print(pos)  # 1

# Không tìm thấy → ValueError
```

```
try:
    pos = fruits.index("grape")
except ValueError:
    print("Không tìm thấy!")

# Tìm trong khoảng
pos = fruits.index("banana", 2)  # Tìm từ index 2
print(pos)  # 3
```

`.count()` - Đếm số lần xuất hiện

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

count = numbers.count(2)
print(count)  # 3

# Không có → trả về 0
count = numbers.count(10)
print(count)  # 0
```

`in` operator - Kiểm tra tồn tại

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

if "banana" in fruits:
    print("Có banana!")

if "grape" not in fruits:
    print("Không có grape!")

# Ứng dụng: Validate input
valid_choices = ["yes", "no", "maybe"]
user_input = input("Chọn (yes/no/maybe): ").lower()

if user_input in valid_choices:
    print(f"Bạn chọn: {user_input}")
else:
    print("Lựa chọn không hợp lệ!")
```

5.4. Sắp Xếp

`.sort()` - Sắp xếp tại chỗ (in-place)

```
numbers = [3, 1, 4, 1, 5, 9, 2]

# Sắp xếp tăng dần
```

```
numbers.sort()
print(numbers)  # [1, 1, 2, 3, 4, 5, 9]

# Sắp xếp giảm dần
numbers.sort(reverse=True)
print(numbers)  # [9, 5, 4, 3, 2, 1, 1]

# Sắp xếp chuỗi
names = ["Chi", "An", "Bình"]
names.sort()
print(names)  # ['An', 'Bình', 'Chi']
```

Sắp xếp với key function:

```
# Sắp xếp theo độ dài chuỗi
words = ["apple", "pie", "banana", "cherry"]
words.sort(key=len)
print(words)  # ['pie', 'apple', 'banana', 'cherry']

# Sắp xếp theo chữ cái cuối
words.sort(key=lambda x: x[-1])
print(words)  # ['banana', 'apple', 'pie', 'cherry']

# Sắp xếp tuple theo phần tử thứ 2
students = [("An", 8), ("Bình", 9), ("Chi", 7)]
students.sort(key=lambda x: x[1], reverse=True)
print(students)  # [('Bình', 9), ('An', 8), ('Chi', 7)]
```

`sorted()` **- Trả về list mới**

```
numbers = [3, 1, 4, 1, 5, 9, 2]

# sorted() trả về list mới
sorted_numbers = sorted(numbers)
print(sorted_numbers)  # [1, 1, 2, 3, 4, 5, 9]
print(numbers)         # [3, 1, 4, 1, 5, 9, 2] - không thay đổi

# Sắp xếp giảm dần
sorted_desc = sorted(numbers, reverse=True)
print(sorted_desc)  # [9, 5, 4, 3, 2, 1, 1]
```

So sánh `.sort()` **vs** `sorted()`**:**

Tiêu chí	`.sort()`	`sorted()`
Thay đổi list gốc	☑ Có	✗ Không
Return value	None	List mới

Tiêu chí	`.sort()`	`sorted()`
Chỉ dùng với list	☑	❌ (dùng với mọi iterable)
Hiệu suất	Nhanh hơn	Chậm hơn (tạo bản copy)

```
# Sai
numbers = [3, 1, 4]
sorted_numbers = numbers.sort()
print(sorted_numbers)  # None (sort() trả về None!)

# Đúng
numbers = [3, 1, 4]
numbers.sort()  # Thay đổi tại chỗ
print(numbers)  # [1, 3, 4]

# Hoặc
numbers = [3, 1, 4]
sorted_numbers = sorted(numbers)  # Tạo list mới
print(sorted_numbers)  # [1, 3, 4]
```

5.5. Đảo Ngược

`.reverse()` - Đảo ngược tại chỗ

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

numbers.reverse()
print(numbers)  # [5, 4, 3, 2, 1]
```

`reversed()` - Trả về iterator

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

# reversed() trả về iterator, cần convert sang list
reversed_numbers = list(reversed(numbers))
print(reversed_numbers)  # [5, 4, 3, 2, 1]
print(numbers)           # [1, 2, 3, 4, 5] - không đổi
```

Slicing `[::-1]` - Tạo list mới

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

reversed_numbers = numbers[::-1]
```

```
print(reversed_numbers)  # [5, 4, 3, 2, 1]
print(numbers)           # [1, 2, 3, 4, 5] - không đổi
```

5.6. Copy List

⚠ Assignment không tạo copy

```
list1 = [1, 2, 3]
list2 = list1  # Không phải copy, chỉ là reference

list2[0] = 100
print(list1)  # [100, 2, 3] - list1 cũng thay đổi!
print(list2)  # [100, 2, 3]

# Giải thích: list1 và list2 cùng trỏ đến 1 object
print(id(list1))  # Cùng ID
print(id(list2))  # Cùng ID
```

Shallow Copy

```
# Cách 1: .copy() method
list1 = [1, 2, 3]
list2 = list1.copy()
list2[0] = 100
print(list1)  # [1, 2, 3] - không đổi
print(list2)  # [100, 2, 3]

# Cách 2: list() constructor
list2 = list(list1)

# Cách 3: Slicing
list2 = list1[:]

# Cách 4: copy module
import copy
list2 = copy.copy(list1)
```

Deep Copy (với nested list)

```
# Shallow copy với nested list
list1 = [[1, 2], [3, 4]]
list2 = list1.copy()

list2[0][0] = 100
print(list1)  # [[100, 2], [3, 4]] - vẫn thay đổi!
print(list2)  # [[100, 2], [3, 4]]
```

```
# Deep copy - copy toàn bộ
import copy
list1 = [[1, 2], [3, 4]]
list2 = copy.deepcopy(list1)

list2[0][0] = 100
print(list1)  # [[1, 2], [3, 4]] - không đổi
print(list2)  # [[100, 2], [3, 4]]
```

---

6. List Operations

6.1. Concatenation (Nối)

```
# Toán tử +
list1 = [1, 2, 3]
list2 = [4, 5, 6]
combined = list1 + list2
print(combined)  # [1, 2, 3, 4, 5, 6]

# Nối nhiều list
result = [1, 2] + [3, 4] + [5, 6]
print(result)  # [1, 2, 3, 4, 5, 6]
```

6.2. Repetition (Lặp)

```
# Toán tử *
numbers = [1, 2, 3]
repeated = numbers * 3
print(repeated)  # [1, 2, 3, 1, 2, 3, 1, 2, 3]

# Tạo list với giá trị khởi tạo
zeros = [0] * 10
print(zeros)  # [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

# ⚠ Cẩn thận với nested list
matrix = [[0] * 3] * 3  # ❌ SAI
matrix[0][0] = 1
print(matrix)  # [[1, 0, 0], [1, 0, 0], [1, 0, 0]] - Tất cả đều đổi!

# ☑ ĐÚNG - Dùng list comprehension
matrix = [[0] * 3 for _ in range(3)]
matrix[0][0] = 1
print(matrix)  # [[1, 0, 0], [0, 0, 0], [0, 0, 0]]
```

6.3. Membership Test

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

# in
print("apple" in fruits)    # True
print("grape" in fruits)    # False

# not in
print("grape" not in fruits)  # True
```

6.4. Length

```
fruits = ["apple", "banana", "orange"]
print(len(fruits))  # 3

# Kiểm tra list rỗng
if len(fruits) == 0:
    print("List rỗng")

# Cách Python hơn
if not fruits:  # Empty list = False
    print("List rỗng")

if fruits:  # Non-empty list = True
    print("List có phần tử")
```

---

7. List Comprehension

7.1. Cú Pháp Cơ Bản

🤯 Đặt vấn đề

Tạo list bình phương từ 0 đến 9.

```
# ❌ Cách thông thường - dài dòng
squares = []
for x in range(10):
    squares.append(x ** 2)
print(squares)  # [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

💡 List Comprehension

```
# ☑ List comprehension - ngắn gọn
squares = [x**2 for x in range(10)]
print(squares)  # [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Cú pháp:

```
[expression for item in iterable]
```

Ví dụ:

```
# Danh sách số từ 0-9
numbers = [x for x in range(10)]

# Danh sách số nhân 2
doubled = [x * 2 for x in range(5)]
print(doubled)  # [0, 2, 4, 6, 8]

# Chuyển sang chữ hoa
names = ["an", "bình", "chi"]
upper_names = [name.upper() for name in names]
print(upper_names)  # ['AN', 'BÌNH', 'CHI']

# Lấy độ dài mỗi từ
words = ["apple", "pie", "banana"]
lengths = [len(word) for word in words]
print(lengths)  # [5, 3, 6]
```

7.2. List Comprehension với Điều Kiện

Filter (if)

```
# Lấy số chẵn
numbers = [x for x in range(10) if x % 2 == 0]
print(numbers)  # [0, 2, 4, 6, 8]

# Lấy số dương
numbers = [-2, -1, 0, 1, 2, 3]
positives = [x for x in numbers if x > 0]
print(positives)  # [1, 2, 3]

# Lọc chuỗi dài hơn 5 ký tự
words = ["apple", "pie", "banana", "cherry"]
long_words = [word for word in words if len(word) > 5]
print(long_words)  # ['banana', 'cherry']
```

Conditional Expression (if-else)

```
# Chẵn → nhân 2, lẻ → nhân 3
numbers = [x*2 if x % 2 == 0 else x*3 for x in range(10)]
print(numbers)  # [0, 3, 4, 9, 8, 15, 12, 21, 16, 27]

# Phân loại điểm
scores = [85, 92, 78, 90, 65]
grades = ["Pass" if s >= 80 else "Fail" for s in scores]
print(grades)  # ['Pass', 'Pass', 'Fail', 'Pass', 'Fail']
```

So sánh vị trí `if`:

```
# Filter: if ở cuối (không có else)
[x for x in range(10) if x % 2 == 0]

# Conditional: if-else ở giữa
[x*2 if x % 2 == 0 else x*3 for x in range(10)]
```

7.3. Nested List Comprehension

```
# Tạo ma trận 3x3
matrix = [[i*3 + j for j in range(3)] for i in range(3)]
print(matrix)
# [[0, 1, 2],
#  [3, 4, 5],
#  [6, 7, 8]]

# Flatten nested list
matrix = [[1, 2], [3, 4], [5, 6]]
flat = [item for row in matrix for item in row]
print(flat)  # [1, 2, 3, 4, 5, 6]

# Tương đương với:
flat = []
for row in matrix:
    for item in row:
        flat.append(item)
```

7.4. List Comprehension vs Map/Filter

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

# Map - áp dụng function
# map()
squared_map = list(map(lambda x: x**2, numbers))

# List comprehension
```

```
squared_comp = [x**2 for x in numbers]

# Filter - lọc phần tử
# filter()
evens_filter = list(filter(lambda x: x % 2 == 0, numbers))

# List comprehension
evens_comp = [x for x in numbers if x % 2 == 0]

# Kết hợp map và filter
# Traditional
result = list(map(lambda x: x**2, filter(lambda x: x % 2 == 0, numbers)))

# List comprehension (dễ đọc hơn)
result = [x**2 for x in numbers if x % 2 == 0]
print(result)  # [4, 16]
```

Khuyến nghị: Dùng list comprehension vì dễ đọc và Pythonic hơn!

---

8. Nested Lists (Lists 2D)

8.1. Tạo Nested Lists

```
# Ma trận 2x3
matrix = [
    [1, 2, 3],
    [4, 5, 6]
]

# Hoặc
matrix = [[1, 2, 3], [4, 5, 6]]

# Tạo ma trận bằng list comprehension
rows, cols = 3, 4
matrix = [[0 for _ in range(cols)] for _ in range(rows)]
print(matrix)
# [[0, 0, 0, 0],
#  [0, 0, 0, 0],
#  [0, 0, 0, 0]]
```

8.2. Truy Cập Elements

```
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
```

```
# Truy cập hàng
print(matrix[0])      # [1, 2, 3]
print(matrix[1])      # [4, 5, 6]

# Truy cập phần tử cụ thể
print(matrix[0][0])   # 1 (hàng 0, cột 0)
print(matrix[1][2])   # 6 (hàng 1, cột 2)
print(matrix[2][1])   # 8 (hàng 2, cột 1)

# Negative indexing
print(matrix[-1][-1]) # 9 (phần tử cuối cùng)
```

8.3. Duyệt Nested Lists

```
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

# Duyệt từng hàng
for row in matrix:
    print(row)

# Duyệt từng phần tử
for row in matrix:
    for element in row:
        print(element, end=" ")
    print()

# Duyệt với index
for i in range(len(matrix)):
    for j in range(len(matrix[i])):
        print(f"matrix[{i}][{j}] = {matrix[i][j]}")

# Duyệt với enumerate
for i, row in enumerate(matrix):
    for j, element in enumerate(row):
        print(f"({i},{j}): {element}")
```

8.4. Ứng Dụng: Tic-Tac-Toe Board

```
# Tạo bàn cờ 3x3
board = [[" " for _ in range(3)] for _ in range(3)]

def print_board(board):
    """In bàn cờ"""
    for row in board:
        print("|".join(row))
```

```
        print("-" * 5)

# Đánh dấu
board[0][0] = "X"
board[1][1] = "O"
board[2][2] = "X"

print_board(board)
# X| |
# -----
#  |O|
# -----
#  | |X
# -----
```

8.5. Thao Tác với Ma Trận

```
# Chuyển vị ma trận (transpose)
matrix = [
    [1, 2, 3],
    [4, 5, 6]
]

# List comprehension
transposed = [[row[i] for row in matrix] for i in range(len(matrix[0]))]
print(transposed)
# [[1, 4],
#  [2, 5],
#  [3, 6]]

# Hoặc dùng zip
transposed = [list(row) for row in zip(*matrix)]
print(transposed)

# Tính tổng từng hàng
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

row_sums = [sum(row) for row in matrix]
print(row_sums)  # [6, 15, 24]

# Tính tổng từng cột
col_sums = [sum(row[i] for row in matrix) for i in range(len(matrix[0]))]
print(col_sums)  # [12, 15, 18]
```

---

9. List Performance và Best Practices

9.1. Time Complexity

Operation	Time Complexity
`list[i]`	O(1)
`list.append(x)`	O(1) amortized
`list.insert(0, x)`	O(n)
`list.pop()`	O(1)
`list.pop(0)`	O(n)
`x in list`	O(n)
`list.sort()`	O(n log n)

9.2. Best Practices

☑ DO

```
# Dùng list comprehension
squares = [x**2 for x in range(10)]

# Dùng enumerate() thay vì range(len())
for i, value in enumerate(my_list):
    print(f"{i}: {value}")

# Kiểm tra empty list
if not my_list:  # Pythonic
    print("Empty")

# Tạo copy an toàn
list2 = list1.copy()

# Dùng sorted() nếu cần giữ list gốc
sorted_list = sorted(original_list)
```

❌ DON'T

```
# Tránh thay đổi list trong vòng lặp
for item in my_list:
    if condition:
        my_list.remove(item)  # ❌ Nguy hiểm!

# Tránh tạo nested list bằng *
matrix = [[0] * 3] * 3  # ❌ Sai

# Tránh append trong vòng lặp khi có list comprehension
result = []
```

```
for x in range(10):
    result.append(x**2)  # ❌ Dùng comprehension thay thế
```

---

10. Bài Tập Thực Hành

Bài 1: Tìm max, min, sum

```
def analyze_list(numbers):
    """
    Phân tích list số

    Returns:
        tuple: (max, min, sum, average)
    """
    if not numbers:
        return None, None, 0, 0

    max_num = max(numbers)
    min_num = min(numbers)
    total = sum(numbers)
    avg = total / len(numbers)

    return max_num, min_num, total, avg

# Test
numbers = [85, 92, 78, 90, 88]
max_n, min_n, total, avg = analyze_list(numbers)
print(f"Max: {max_n}, Min: {min_n}, Sum: {total}, Avg: {avg:.2f}")
# Max: 92, Min: 78, Sum: 433, Avg: 86.60
```

Bài 2: Xóa phần tử trùng lặp

```
def remove_duplicates(items):
    """
    Xóa phần tử trùng lặp (giữ thứ tự)

    Args:
        items (list): List gốc

    Returns:
        list: List không trùng lặp
    """
    seen = []
    for item in items:
        if item not in seen:
            seen.append(item)
    return seen
```

```python
# Cách 2: Dùng dict (Python 3.7+, dict giữ thứ tự)
def remove_duplicates_v2(items):
    return list(dict.fromkeys(items))

# Cách 3: Set (không giữ thứ tự)
def remove_duplicates_v3(items):
    return list(set(items))

# Test
numbers = [1, 2, 3, 2, 4, 1, 5]
print(remove_duplicates(numbers))  # [1, 2, 3, 4, 5]
```

Bài 3: Tìm phần tử xuất hiện nhiều nhất

```python
def most_common(items):
    """
    Tìm phần tử xuất hiện nhiều nhất

    Returns:
        tuple: (element, count)
    """
    if not items:
        return None, 0

    max_count = 0
    max_item = None

    for item in items:
        count = items.count(item)
        if count > max_count:
            max_count = count
            max_item = item

    return max_item, max_count

# Cách 2: Dùng Counter
from collections import Counter

def most_common_v2(items):
    if not items:
        return None, 0
    counter = Counter(items)
    return counter.most_common(1)[0]

# Test
numbers = [1, 2, 3, 2, 4, 2, 5]
item, count = most_common(numbers)
print(f"Phần tử xuất hiện nhiều nhất: {item} ({count} lần)")
# Phần tử xuất hiện nhiều nhất: 2 (3 lần)
```

Bài 4: Merge hai list đã sắp xếp

```
def merge_sorted_lists(list1, list2):
    """
    Merge 2 list đã sắp xếp thành 1 list sắp xếp

    Args:
        list1, list2: Hai list đã sắp xếp tăng dần

    Returns:
        list: List đã merge và sắp xếp
    """
    result = []
    i, j = 0, 0

    while i < len(list1) and j < len(list2):
        if list1[i] <= list2[j]:
            result.append(list1[i])
            i += 1
        else:
            result.append(list2[j])
            j += 1

    # Thêm phần còn lại
    result.extend(list1[i:])
    result.extend(list2[j:])

    return result

# Test
list1 = [1, 3, 5, 7]
list2 = [2, 4, 6, 8]
print(merge_sorted_lists(list1, list2))
# [1, 2, 3, 4, 5, 6, 7, 8]
```

Bài 5: Rotate list

```
def rotate_list(items, k):
    """
    Xoay list sang phải k vị trí

    Args:
        items (list): List gốc
        k (int): Số vị trí xoay

    Returns:
        list: List sau khi xoay
```

```
    """
    if not items or k == 0:
        return items

    k = k % len(items)  # Xử lý k lớn hơn độ dài
    return items[-k:] + items[:-k]

# Test
numbers = [1, 2, 3, 4, 5]
print(rotate_list(numbers, 2))  # [4, 5, 1, 2, 3]
print(rotate_list(numbers, 7))  # [4, 5, 1, 2, 3] (7 % 5 = 2)
```

Bài 6: Tìm tất cả cặp có tổng = target

```
def find_pairs_with_sum(numbers, target):
    """
    Tìm tất cả cặp số có tổng = target

    Args:
        numbers (list): Danh sách số
        target (int): Tổng cần tìm

    Returns:
        list: Danh sách các cặp
    """
    pairs = []
    seen = set()

    for num in numbers:
        complement = target - num
        if complement in seen:
            pairs.append((complement, num))
        seen.add(num)

    return pairs

# Test
numbers = [2, 7, 11, 15, 3, 6]
target = 9
print(find_pairs_with_sum(numbers, target))
# [(2, 7), (3, 6)]
```

Bài 7: Flatten nested list

```
def flatten(nested_list):
    """
    Làm phẳng nested list (bất kỳ độ sâu)
```

```
    Args:
        nested_list: List có thể chứa nested lists

    Returns:
        list: List đã flatten
    """
    result = []

    for item in nested_list:
        if isinstance(item, list):
            result.extend(flatten(item))  # Recursive
        else:
            result.append(item)

    return result

# Test
nested = [1, [2, 3], [4, [5, 6]], 7]
print(flatten(nested))  # [1, 2, 3, 4, 5, 6, 7]
```

Bài 8: Tạo ma trận xoắn ốc

```
def spiral_matrix(n):
    """
    Tạo ma trận xoắn ốc n x n

    Example:
        n = 3 → [[1, 2, 3],
                 [8, 9, 4],
                 [7, 6, 5]]
    """
    matrix = [[0] * n for _ in range(n)]

    num = 1
    top, bottom = 0, n - 1
    left, right = 0, n - 1

    while top <= bottom and left <= right:
        # Đi sang phải
        for i in range(left, right + 1):
            matrix[top][i] = num
            num += 1
        top += 1

        # Đi xuống
        for i in range(top, bottom + 1):
            matrix[i][right] = num
            num += 1
        right -= 1
```

```
        # Đi sang trái
        if top <= bottom:
            for i in range(right, left - 1, -1):
                matrix[bottom][i] = num
                num += 1
            bottom -= 1

        # Đi lên
        if left <= right:
            for i in range(bottom, top - 1, -1):
                matrix[i][left] = num
                num += 1
            left += 1

    return matrix

# Test
matrix = spiral_matrix(4)
for row in matrix:
    print(row)
# [1, 2, 3, 4]
# [12, 13, 14, 5]
# [11, 16, 15, 6]
# [10, 9, 8, 7]
```

---

11. Tổng Kết và Checklist

☑ Kiến thức cần nắm vững

Cơ bản:

- ☐ Tạo list: literal, `list()`, `range()`
- ☐ Indexing và slicing
- ☐ Thay đổi elements (mutable)
- ☐ `len()`, `in`, `not in`

Methods:

- ☐ Thêm: `append()`, `extend()`, `insert()`
- ☐ Xóa: `remove()`, `pop()`, `clear()`, `del`
- ☐ Tìm: `index()`, `count()`
- ☐ Sắp xếp: `sort()`, `sorted()`, `reverse()`
- ☐ Copy: `copy()`, shallow vs deep copy

Nâng cao:

- ☐ List comprehension cơ bản
- ☐ List comprehension với điều kiện
- ☐ Nested list comprehension

- ☐ Nested lists (2D arrays)
- ☐ Operations: `+`, `*`, slicing

🎯 Lưu ý quan trọng

1. **List là mutable** - có thể thay đổi sau khi tạo
2. **Assignment không tạo copy** - dùng `.copy()` hoặc `[:]`
3. **`.sort()` thay đổi list gốc**, `sorted()` tạo list mới
4. **List comprehension** nhanh hơn và Pythonic hơn vòng lặp
5. **Cẩn thận với nested list** khi dùng `*`
6. **`.append()` vs `.extend()`** - append thêm 1, extend thêm nhiều
7. **Time complexity** - insert/pop đầu list chậm O(n)

📚 Bước tiếp theo

Sau khi nắm vững Lists, bạn sẵn sàng học:

- **Tuples** - list không thay đổi được
- **Sets** - collection không trùng lặp
- **Dictionaries** - key-value pairs
- **Advanced comprehensions** - dict/set comprehension

---

12. Câu Hỏi Thường Gặp (FAQ)

Q1: Tại sao `list2 = list1` không tạo copy?

A: Vì Python dùng **reference**. `list2` chỉ trỏ đến cùng object với `list1`.

```
list1 = [1, 2, 3]
list2 = list1  # Reference, không phải copy

list2[0] = 100
print(list1)  # [100, 2, 3] - list1 cũng đổi!

# Để copy:
list2 = list1.copy()  # hoặc list1[:]
```

Q2: Khi nào dùng `.sort()` vs `sorted()`?

A:

- `.sort()`: Khi muốn sắp xếp **tại chỗ**, không cần list gốc
- `sorted()`: Khi muốn **giữ list gốc** và tạo bản copy đã sắp xếp

```
numbers = [3, 1, 4]

# sort() - không return gì
```

```
numbers.sort()
print(numbers)  # [1, 3, 4]

# sorted() - return list mới
numbers = [3, 1, 4]
sorted_nums = sorted(numbers)
print(numbers)      # [3, 1, 4] - không đổi
print(sorted_nums)  # [1, 3, 4]
```

Q3: Làm sao để xóa phần tử trong vòng lặp?

A: Không nên xóa trực tiếp. Tạo list mới hoặc dùng list comprehension.

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

# ❌ SAI - skip phần tử
for num in numbers:
    if num % 2 == 0:
        numbers.remove(num)  # Nguy hiểm!

# ☑ ĐÚNG - List comprehension
numbers = [1, 2, 3, 4, 5]
numbers = [num for num in numbers if num % 2 != 0]
print(numbers)  # [1, 3, 5]

# ☑ ĐÚNG - Duyệt ngược
numbers = [1, 2, 3, 4, 5]
for i in range(len(numbers) - 1, -1, -1):
    if numbers[i] % 2 == 0:
        numbers.pop(i)
```

Q4: Sự khác biệt giữa `append()` và `extend()`?

A:

- `append()`: Thêm **1 phần tử** (có thể là list)
- `extend()`: Thêm **nhiều phần tử** từ iterable

```
list1 = [1, 2]
list1.append([3, 4])
print(list1)  # [1, 2, [3, 4]] - thêm list làm 1 phần tử

list2 = [1, 2]
list2.extend([3, 4])
print(list2)  # [1, 2, 3, 4] - thêm từng phần tử
```

Q5: List comprehension có nhanh hơn vòng lặp?

A: Có, vì list comprehension được tối ưu ở C level.

```
import timeit

# Vòng lặp
def with_loop():
    result = []
    for x in range(1000):
        result.append(x**2)
    return result

# List comprehension
def with_comp():
    return [x**2 for x in range(1000)]

print(timeit.timeit(with_loop, number=10000))  # ~0.5s
print(timeit.timeit(with_comp, number=10000))  # ~0.4s

# Comprehension nhanh hơn ~20-30%
```

---

13. Thử Thách Cuối Khóa

Challenge 1: Two Sum

Cho list số và target, tìm 2 số có tổng = target.

```
"""
Input: nums = [2, 7, 11, 15], target = 9
Output: [0, 1] (vì nums[0] + nums[1] = 9)

Yêu cầu: O(n) time complexity
Gợi ý: Dùng dictionary để lưu complement
"""
```

Challenge 2: Move Zeros

Di chuyển tất cả số 0 về cuối list, giữ thứ tự các số khác 0.

```
"""
Input: [0, 1, 0, 3, 12]
Output: [1, 3, 12, 0, 0]

Yêu cầu: In-place (không tạo list mới)
"""
```

Challenge 3: Product Except Self

Cho list số, trả về list mà mỗi phần tử = tích tất cả số khác nó.

```
"""
Input: [1, 2, 3, 4]
Output: [24, 12, 8, 6]
Giải thích:
- Index 0: 2 * 3 * 4 = 24
- Index 1: 1 * 3 * 4 = 12
- Index 2: 1 * 2 * 4 = 8
- Index 3: 1 * 2 * 3 = 6

Yêu cầu: Không dùng phép chia
"""
```

Challenge 4: Pascal's Triangle

Sinh n hàng đầu tiên của tam giác Pascal.

```
"""
Input: n = 5
Output:
[
     [1],
    [1,1],
   [1,2,1],
  [1,3,3,1],
 [1,4,6,4,1]
]

Quy tắc: Mỗi số = tổng 2 số phía trên
"""
```
