Python Programming - Sets (Tập Hợp)

> **Mục tiêu học tập**: Làm chủ Sets - cấu trúc dữ liệu tập hợp không trùng lặp với các phép toán tập hợp mạnh mẽ (union, intersection, difference).

1. Giới Thiệu về Sets

1.1. Set là gì?

😨 Đặt vấn đề

Giả sử bạn cần lưu danh sách email của người đăng ký, nhưng mỗi email chỉ tính 1 lần.

```
# ❌ Dùng list - có thể trùng lặp
emails = ["user1@gmail.com", "user2@gmail.com", "user1@gmail.com"]
print(len(emails))  # 3 - nhưng chỉ có 2 email unique!

# Phải tự kiểm tra trùng
unique_emails = []
for email in emails:
    if email not in unique_emails:
        unique_emails.append(email)
print(len(unique_emails))  # 2
```

Làm sao để tự động loại bỏ phần tử trùng lặp?

💡 Giải quyết: Sets

Set là cấu trúc dữ liệu **tập hợp không trùng lặp**.

```
# ☑ Dùng set - tự động loại bỏ trùng lặp
emails = {"user1@gmail.com", "user2@gmail.com", "user1@gmail.com"}
print(emails)  # {'user1@gmail.com', 'user2@gmail.com'}
print(len(emails))  # 2 - tự động unique!
```

Đặc điểm của Set:

- ☑ **Unordered** (không có thứ tự): Không thể truy cập bằng index
- ☑ **No duplicates** (không trùng lặp): Mỗi phần tử chỉ xuất hiện 1 lần
- ☑ **Mutable** (có thể thay đổi): Có thể thêm/xóa phần tử
- ☑ **Elements phải immutable**: Chỉ chứa string, number, tuple (không chứa list/dict)
- ☑ **Fast membership test**: O(1) kiểm tra phần tử tồn tại
- ☑ **Set operations**: union, intersection, difference

1.2. Khi Nào Dùng Set?

Dùng Set khi:

- Cần loại bỏ trùng lặp
- Cần kiểm tra membership nhanh (`in` operator)
- Cần các phép toán tập hợp (giao, hợp, hiệu)
- Không quan tâm đến thứ tự

Dùng List khi:

- Cần giữ thứ tự
- Cho phép trùng lặp
- Cần truy cập bằng index
- Cần slicing

So sánh:

Tiêu chí	Set	List	Dict
Ordered	❌	☑	☑ (3.7+)
Duplicates	❌	☑	Keys: ❌, Values: ☑
Indexing	❌	☑	Key-based
Mutable	☑	☑	☑
Use case	Unique items	Sequences	Key-value mapping

---

2. Tạo Sets

2.1. Set Literals

```
# Set với curly braces {}
fruits = {"apple", "banana", "orange"}
print(fruits)  # {'apple', 'banana', 'orange'}

# Set số
numbers = {1, 2, 3, 4, 5}
print(numbers)  # {1, 2, 3, 4, 5}

# Set hỗn hợp
mixed = {1, "two", 3.0, True}
print(mixed)  # {1, 'two', 3.0} - True == 1 nên chỉ giữ 1

# ⚠ Set rỗng phải dùng set(), không phải {}
empty_set = set()  # ☑ Set rỗng
empty_dict = {}    # ❌ Dictionary rỗng, không phải set!

print(type(empty_set))   # <class 'set'>
print(type(empty_dict))  # <class 'dict'>
```

2.2. Hàm `set()`

Chuyển đổi iterable thành set (tự động loại bỏ trùng lặp).

```
# Từ list - loại bỏ trùng lặp
numbers = [1, 2, 3, 2, 4, 1, 5]
unique_numbers = set(numbers)
print(unique_numbers)  # {1, 2, 3, 4, 5}

# Từ string - mỗi ký tự là 1 phần tử
chars = set("hello")
print(chars)  # {'h', 'e', 'l', 'o'} - 'l' chỉ xuất hiện 1 lần

# Từ tuple
coordinates = set((1, 2, 3, 2, 1))
print(coordinates)  # {1, 2, 3}

# Từ range
numbers = set(range(5))
print(numbers)  # {0, 1, 2, 3, 4}
```

2.3. Set Comprehension

```
# Tạo set bình phương
squares = {x**2 for x in range(5)}
print(squares)  # {0, 1, 4, 9, 16}

# Với điều kiện
evens = {x for x in range(10) if x % 2 == 0}
print(evens)  # {0, 2, 4, 6, 8}

# Từ string - chữ hoa
text = "Hello World"
uppercase = {char.upper() for char in text if char.isalpha()}
print(uppercase)  # {'H', 'E', 'L', 'O', 'W', 'R', 'D'}
```

Ứng dụng: Unique words trong văn bản

```
text = "apple banana apple cherry banana apple"
unique_words = set(text.split())
print(unique_words)  # {'banana', 'apple', 'cherry'}
print(f"Số từ unique: {len(unique_words)}")  # 3
```

---

3. Truy Cập Elements

3.1. Không Có Indexing

```
fruits = {"apple", "banana", "orange"}

# ❌ Không thể truy cập bằng index
# print(fruits[0])  # TypeError: 'set' object is not subscriptable

# ❌ Không có slicing
# print(fruits[0:2])  # TypeError
```

3.2. Kiểm Tra Membership

```
fruits = {"apple", "banana", "orange"}

# Kiểm tra phần tử tồn tại (rất nhanh - O(1))
if "apple" in fruits:
    print("Có apple!")  # Có apple!

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

# Ứng dụng: Validate input
valid_commands = {"start", "stop", "restart", "status"}
user_input = input("Nhập lệnh: ").lower()

if user_input in valid_commands:
    print(f"Thực thi: {user_input}")
else:
    print("Lệnh không hợp lệ!")
```

3.3. Duyệt Qua Set

```
fruits = {"apple", "banana", "orange"}

# Duyệt với for (thứ tự không đảm bảo)
for fruit in fruits:
    print(fruit)

# Với enumerate
for i, fruit in enumerate(fruits, 1):
    print(f"{i}. {fruit}")
```

---

4. Thêm và Xóa Elements

4.1. Thêm Elements

`.add()` - Thêm 1 phần tử

```
fruits = {"apple", "banana"}

# Thêm 1 phần tử
fruits.add("orange")
print(fruits)  # {'apple', 'banana', 'orange'}

# Thêm phần tử đã tồn tại → không thay đổi
fruits.add("apple")
print(fruits)  # {'apple', 'banana', 'orange'} - không trùng
```

`.update()` - Thêm nhiều phần tử

```
fruits = {"apple", "banana"}

# Update từ list
fruits.update(["orange", "grape"])
print(fruits)  # {'apple', 'banana', 'orange', 'grape'}

# Update từ set khác
fruits.update({"kiwi", "mango"})
print(fruits)  # {'apple', 'banana', 'orange', 'grape', 'kiwi', 'mango'}

# Update từ nhiều iterables
fruits.update(["cherry"], {"peach"}, ("plum",))
print(fruits)

# Tự động loại bỏ trùng lặp
numbers = {1, 2, 3}
numbers.update([3, 4, 5])
print(numbers)  # {1, 2, 3, 4, 5} - 3 không bị trùng
```

4.2. Xóa Elements

`.remove()` - Xóa phần tử (lỗi nếu không tồn tại)

```
fruits = {"apple", "banana", "orange"}

# Xóa phần tử tồn tại
fruits.remove("banana")
print(fruits)  # {'apple', 'orange'}

# ❌ Xóa phần tử không tồn tại → KeyError
try:
    fruits.remove("grape")
```

```
except KeyError:
    print("Phần tử không tồn tại!")
```

`.discard()` - Xóa phần tử (không lỗi nếu không tồn tại)

```
fruits = {"apple", "banana", "orange"}

# Xóa phần tử tồn tại
fruits.discard("banana")
print(fruits)  # {'apple', 'orange'}

# ☑ Xóa phần tử không tồn tại → không lỗi
fruits.discard("grape")  # OK, không làm gì
print(fruits)  # {'apple', 'orange'}
```

Khi nào dùng `.remove()` vs `.discard()`?

- `.remove()`: Khi **chắc chắn** phần tử tồn tại, muốn lỗi nếu không có
- `.discard()`: Khi **không chắc** phần tử có hay không, muốn an toàn

`.pop()` - Xóa phần tử ngẫu nhiên

```
fruits = {"apple", "banana", "orange"}

# Pop phần tử ngẫu nhiên (vì set không có thứ tự)
removed = fruits.pop()
print(f"Removed: {removed}")
print(f"Remaining: {fruits}")

# ⚠ Set rỗng → KeyError
empty_set = set()
try:
    empty_set.pop()
except KeyError:
    print("Set rỗng!")
```

`.clear()` - Xóa tất cả

```
fruits = {"apple", "banana", "orange"}

fruits.clear()
print(fruits)  # set() - set rỗng
```

---

5. Set Operations (Phép Toán Tập Hợp)

5.1. Union (Hợp) - | hoặc .union()

Định nghĩa: Tất cả phần tử từ cả hai sets (không trùng lặp).

```
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}

# Cách 1: Toán tử |
union = A | B
print(union)  # {1, 2, 3, 4, 5, 6}

# Cách 2: Method .union()
union = A.union(B)
print(union)  # {1, 2, 3, 4, 5, 6}

# Union nhiều sets
C = {7, 8}
union = A | B | C
print(union)  # {1, 2, 3, 4, 5, 6, 7, 8}

# .union() với nhiều iterables
union = A.union(B, [9, 10], {11, 12})
print(union)  # {1, 2, 3, 4, 5, 6, 9, 10, 11, 12}
```

Ví dụ thực tế:

```
# Học sinh tham gia CLB
math_club = {"An", "Bình", "Chi"}
sports_club = {"Bình", "Dũng", "Em"}

# Tất cả học sinh tham gia ít nhất 1 CLB
all_students = math_club | sports_club
print(all_students)  # {'An', 'Bình', 'Chi', 'Dũng', 'Em'}
```

5.2. Intersection (Giao) - & hoặc .intersection()

Định nghĩa: Phần tử có trong cả hai sets.

```
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}

# Cách 1: Toán tử &
intersection = A & B
print(intersection)  # {3, 4}

# Cách 2: Method .intersection()
```

```
intersection = A.intersection(B)
print(intersection)  # {3, 4}

# Intersection nhiều sets
C = {2, 3, 7}
intersection = A & B & C
print(intersection)  # {3} - chỉ 3 có trong cả 3 sets
```

Ví dụ thực tế:

```
# Học sinh tham gia CẢ HAI CLB
math_club = {"An", "Bình", "Chi"}
sports_club = {"Bình", "Dũng", "Em"}

# Học sinh tham gia cả 2 CLB
both_clubs = math_club & sports_club
print(both_clubs)  # {'Bình'}
```

5.3. Difference (Hiệu) - `-` hoặc `.difference()`

Định nghĩa: Phần tử có trong A nhưng không có trong B.

```
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}

# Cách 1: Toán tử -
diff = A - B
print(diff)  # {1, 2} - có trong A, không trong B

# Cách 2: Method .difference()
diff = A.difference(B)
print(diff)  # {1, 2}

# ⚠ Không giao hoán: A - B ≠ B - A
diff_BA = B - A
print(diff_BA)  # {5, 6} - có trong B, không trong A
```

Ví dụ thực tế:

```
# Học sinh chỉ tham gia Math CLB
math_club = {"An", "Bình", "Chi"}
sports_club = {"Bình", "Dũng", "Em"}

# Chỉ tham gia Math, không Sports
only_math = math_club - sports_club
print(only_math)  # {'An', 'Chi'}
```

```
# Chỉ tham gia Sports, không Math
only_sports = sports_club - math_club
print(only_sports)  # {'Dũng', 'Em'}
```

5.4. Symmetric Difference (Hiệu Đối Xứng) - `^` hoặc `.symmetric_difference()`

Định nghĩa: Phần tử có trong A hoặc B, nhưng không có trong cả hai.

```
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}

# Cách 1: Toán tử ^
sym_diff = A ^ B
print(sym_diff)  # {1, 2, 5, 6} - không chung

# Cách 2: Method .symmetric_difference()
sym_diff = A.symmetric_difference(B)
print(sym_diff)  # {1, 2, 5, 6}

# Tương đương: (A - B) | (B - A)
sym_diff = (A - B) | (B - A)
print(sym_diff)  # {1, 2, 5, 6}
```

Ví dụ thực tế:

```
# Học sinh chỉ tham gia 1 CLB (không cả 2)
math_club = {"An", "Bình", "Chi"}
sports_club = {"Bình", "Dũng", "Em"}

# Chỉ tham gia 1 CLB
only_one_club = math_club ^ sports_club
print(only_one_club)  # {'An', 'Chi', 'Dũng', 'Em'}
```

5.5. Tóm Tắt Set Operations

Operation	Operator	Method	Ý nghĩa
Union	`A | B`	`A.union(B)`	A hoặc B (tất cả)
Intersection	`A & B`	`A.intersection(B)`	A và B (chung)
Difference	`A - B`	`A.difference(B)`	Có trong A, không trong B
Symmetric Diff	`A ^ B`	`A.symmetric_difference(B)`	Có trong A hoặc B, không cả hai

Diagram:

```
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}

A | B  = {1, 2, 3, 4, 5, 6}  # Union
A & B  = {3, 4}               # Intersection
A - B  = {1, 2}               # Difference (A - B)
B - A  = {5, 6}               # Difference (B - A)
A ^ B  = {1, 2, 5, 6}         # Symmetric Difference
```

---

6. Set Methods Nâng Cao

6.1. Subset và Superset

`.issubset()` - Kiểm tra tập con

```
A = {1, 2, 3}
B = {1, 2, 3, 4, 5}

# A có phải tập con của B?
print(A.issubset(B))  # True (A ⊆ B)

# Toán tử <=
print(A <= B)  # True

# Strict subset: A ⊂ B (A subset của B và A ≠ B)
print(A < B)  # True
print(A < A)  # False (không strict)
```

`.issuperset()` - Kiểm tra tập cha

```
A = {1, 2, 3, 4, 5}
B = {1, 2, 3}

# A có phải tập cha của B?
print(A.issuperset(B))  # True (A ⊇ B)

# Toán tử >=
print(A >= B)  # True

# Strict superset: A ⊃ B
print(A > B)  # True
print(A > A)  # False
```

Ví dụ thực tế:

```
# Kiểm tra học sinh có đủ skills yêu cầu?
required_skills = {"Python", "SQL", "Git"}
candidate_skills = {"Python", "SQL", "Git", "Docker"}

if required_skills.issubset(candidate_skills):
    print("Ứng viên đủ yêu cầu!")
# hoặc
if candidate_skills >= required_skills:
    print("Ứng viên đủ yêu cầu!")
```

6.2. `.isdisjoint()` - Kiểm tra không giao nhau

```
A = {1, 2, 3}
B = {4, 5, 6}
C = {3, 4, 5}

# A và B không có phần tử chung?
print(A.isdisjoint(B))  # True (không giao nhau)
print(A.isdisjoint(C))  # False (có 3 chung)
```

Ví dụ thực tế:

```
# Kiểm tra lịch làm việc không trùng
week1_days = {"Monday", "Tuesday", "Wednesday"}
week2_days = {"Thursday", "Friday", "Saturday"}

if week1_days.isdisjoint(week2_days):
    print("Lịch không trùng!")  # Lịch không trùng!
```

6.3. In-Place Operations

Các operations có thể thay đổi set gốc.

```
A = {1, 2, 3}
B = {3, 4, 5}

# |= : A = A | B
A |= B
print(A)  # {1, 2, 3, 4, 5}

# &= : A = A & B
A = {1, 2, 3}
A &= B
print(A)  # {3}

# -= : A = A - B
```

```
A = {1, 2, 3, 4}
A -= {2, 3}
print(A)  # {1, 4}

# ^= : A = A ^ B
A = {1, 2, 3}
A ^= B
print(A)  # {1, 2, 4, 5}
```

Methods tương ứng:

```
A = {1, 2, 3}
B = {3, 4, 5}

# update() = |=
A.update(B)
print(A)  # {1, 2, 3, 4, 5}

# intersection_update() = &=
A = {1, 2, 3}
A.intersection_update(B)
print(A)  # {3}

# difference_update() = -=
A = {1, 2, 3}
A.difference_update(B)
print(A)  # {1, 2}

# symmetric_difference_update() = ^=
A = {1, 2, 3}
A.symmetric_difference_update(B)
print(A)  # {1, 2, 4, 5}
```

---

7. Frozen Sets

7.1. Tạo Frozen Set

Frozen set là set **không thể thay đổi** (immutable).

```
# Tạo frozenset
numbers = frozenset([1, 2, 3, 4, 5])
print(numbers)  # frozenset({1, 2, 3, 4, 5})

# Từ set
mutable_set = {1, 2, 3}
frozen = frozenset(mutable_set)
print(frozen)  # frozenset({1, 2, 3})
```

7.2. Không Thể Thay Đổi

```
frozen = frozenset([1, 2, 3])

# ❌ Không thể thêm
# frozen.add(4)  # AttributeError

# ❌ Không thể xóa
# frozen.remove(1)  # AttributeError

# ☑ Có thể dùng set operations (tạo frozenset mới)
A = frozenset([1, 2, 3])
B = frozenset([3, 4, 5])

union = A | B
print(union)  # frozenset({1, 2, 3, 4, 5})
print(type(union))  # <class 'frozenset'>
```

7.3. Khi Nào Dùng Frozen Set?

Dùng frozenset khi:

- Cần dùng set làm **key trong dictionary**
- Cần dùng set làm **phần tử của set khác**
- Cần đảm bảo set **không bị thay đổi**

```
# ❌ Set thường không thể làm key
# d = {{1, 2}: "value"}  # TypeError: unhashable type: 'set'

# ☑ Frozenset có thể làm key
d = {frozenset([1, 2]): "value"}
print(d)  # {frozenset({1, 2}): 'value'}

# ☑ Frozenset có thể là phần tử của set
sets_collection = {frozenset([1, 2]), frozenset([3, 4])}
print(sets_collection)  # {frozenset({1, 2}), frozenset({3, 4})}
```

---

8. Ứng Dụng Thực Tế

8.1. Loại Bỏ Trùng Lặp

```
# Từ list
numbers = [1, 2, 3, 2, 4, 1, 5, 3]
unique = list(set(numbers))
print(unique)  # [1, 2, 3, 4, 5] (thứ tự không đảm bảo)
```

```
# Giữ thứ tự với dict.fromkeys() (Python 3.7+)
unique = list(dict.fromkeys(numbers))
print(unique)  # [1, 2, 3, 4, 5] (giữ thứ tự)

# Từ string
text = "hello"
unique_chars = set(text)
print(unique_chars)  # {'h', 'e', 'l', 'o'}
```

8.2. Tìm Common Elements

```
# Tìm học sinh tham gia cả 2 lớp
class_a = {"An", "Bình", "Chi", "Dũng"}
class_b = {"Bình", "Chi", "Em", "Giang"}

# Học sinh trong cả 2 lớp
common = class_a & class_b
print(f"Học sinh cả 2 lớp: {common}")  # {'Bình', 'Chi'}

# Học sinh chỉ ở lớp A
only_a = class_a - class_b
print(f"Chỉ lớp A: {only_a}")  # {'An', 'Dũng'}
```

8.3. Validate Permissions

```
# Kiểm tra user có đủ quyền?
def has_permissions(user_perms, required_perms):
    """
    Kiểm tra user có tất cả quyền yêu cầu

    Args:
        user_perms (set): Quyền của user
        required_perms (set): Quyền yêu cầu

    Returns:
        bool: True nếu đủ quyền
    """
    return required_perms.issubset(user_perms)

# Test
user = {"read", "write", "delete"}
admin_required = {"read", "write", "delete", "admin"}
basic_required = {"read", "write"}

print(has_permissions(user, basic_required))   # True
print(has_permissions(user, admin_required))   # False
```

8.4. Find Missing Items

```
# Tìm missing IDs
expected_ids = set(range(1, 11))  # 1-10
received_ids = {1, 3, 5, 7, 9, 2, 4}

missing = expected_ids - received_ids
print(f"Missing IDs: {missing}")  # {6, 8, 10}

extra = received_ids - expected_ids
print(f"Extra IDs: {extra}")  # set() (không có)
```

8.5. Tag Management

```
# Quản lý tags của bài viết
post1_tags = {"python", "programming", "tutorial"}
post2_tags = {"python", "data-science", "ml"}

# Tags chung
common_tags = post1_tags & post2_tags
print(f"Common tags: {common_tags}")  # {'python'}

# Tất cả tags unique
all_tags = post1_tags | post2_tags
print(f"All tags: {all_tags}")
# {'python', 'programming', 'tutorial', 'data-science', 'ml'}

# Tags chỉ có ở post1
unique_to_post1 = post1_tags - post2_tags
print(f"Unique to post1: {unique_to_post1}")
# {'programming', 'tutorial'}
```

---

9. Bài Tập Thực Hành

Bài 1: Unique Characters in Two Strings

```
def unique_chars(s1, s2):
    """
    Tìm ký tự xuất hiện trong s1 hoặc s2, nhưng không cả hai

    Args:
        s1, s2 (str): Hai chuỗi

    Returns:
        set: Ký tự unique
    """
    set1 = set(s1)
    set2 = set(s2)
    return set1 ^ set2
```

```
# Test
print(unique_chars("hello", "world"))  # {'e', 'h', 'r', 'w', 'd'}
print(unique_chars("abc", "cde"))      # {'a', 'b', 'd', 'e'}
```

Bài 2: Find Common Friends (Bài 2 Assessment tương tự)

```
def common_friends(friend_lists):
    """
    Tìm bạn chung của tất cả mọi người

    Args:
        friend_lists (list of sets): Danh sách bạn bè

    Returns:
        set: Bạn chung
    """
    if not friend_lists:
        return set()

    # Giao của tất cả sets
    common = friend_lists[0]
    for friends in friend_lists[1:]:
        common &= friends

    return common

# Hoặc dùng set.intersection()
def common_friends_v2(friend_lists):
    if not friend_lists:
        return set()
    return set.intersection(*friend_lists)

# Test
alice_friends = {"Bob", "Charlie", "David"}
bob_friends = {"Alice", "Charlie", "Eve"}
charlie_friends = {"Alice", "Bob", "David"}

common = common_friends([alice_friends, bob_friends, charlie_friends])
print(common)  # set() - không có bạn chung cả 3
```

Bài 3: Most Similar Strings (Bài 2 Assessment)

```
def most_similar_strings(strings):
    """
    Tìm 2 chuỗi có nhiều ký tự chung nhất
```

```
    Args:
        strings (list): Danh sách chuỗi

    Returns:
        tuple: (str1, str2, num_common_chars)
    """
    max_common = 0
    result = None

    # So sánh từng cặp
    for i in range(len(strings)):
        for j in range(i + 1, len(strings)):
            s1, s2 = strings[i], strings[j]

            # Ký tự chung
            common = set(s1) & set(s2)
            num_common = len(common)

            if num_common > max_common:
                max_common = num_common
                result = (s1, s2, num_common)

    return result if result else (None, None, 0)

# Test
strings = ["hello", "world", "help", "hero"]
s1, s2, count = most_similar_strings(strings)
print(f"Most similar: '{s1}' and '{s2}' ({count} common chars)")
# Most similar: 'hello' and 'help' (4 common chars)
```

Bài 4: Vowels and Consonants

```
def count_vowels_consonants(text):
    """
    Đếm số nguyên âm và phụ âm trong chuỗi

    Args:
        text (str): Chuỗi cần đếm

    Returns:
        tuple: (num_vowels, num_consonants)
    """
    vowels = set("aeiouAEIOU")
    text_chars = set(char for char in text if char.isalpha())

    vowel_chars = text_chars & vowels
    consonant_chars = text_chars - vowels

    return len(vowel_chars), len(consonant_chars)
```

```
# Test
text = "Hello World"
v, c = count_vowels_consonants(text)
print(f"Vowels: {v}, Consonants: {c}")  # Vowels: 2, Consonants: 6
```

Bài 5: Anagram Groups

```
def group_anagrams(words):
    """
    Nhóm các từ là anagram của nhau

    Args:
        words (list): Danh sách từ

    Returns:
        list of lists: Các nhóm anagram
    """
    from collections import defaultdict

    groups = defaultdict(list)

    for word in words:
        # Key là frozenset của các ký tự
        key = frozenset(word)
        groups[key].append(word)

    return list(groups.values())

# Test
words = ["eat", "tea", "tan", "ate", "nat", "bat"]
groups = group_anagrams(words)
print(groups)
# [['eat', 'tea', 'ate'], ['tan', 'nat'], ['bat']]
```

Bài 6: Set Similarity (Jaccard Index)

```
def jaccard_similarity(set1, set2):
    """
    Tính độ tương đồng Jaccard giữa 2 sets
    Jaccard = |A ∩ B| / |A ∪ B|

    Args:
        set1, set2 (set): Hai sets

    Returns:
        float: Độ tương đồng (0-1)
    """
```

```
    intersection = set1 & set2
    union = set1 | set2

    if not union:
        return 0.0

    return len(intersection) / len(union)

# Test
A = {"apple", "banana", "orange"}
B = {"banana", "orange", "grape", "kiwi"}

similarity = jaccard_similarity(A, B)
print(f"Similarity: {similarity:.2f}")  # 0.40
```

Bài 7: Find Duplicate Rows (2D List)

```
def find_duplicate_rows(matrix):
    """
    Tìm các hàng trùng lặp trong ma trận

    Args:
        matrix (list of lists): Ma trận

    Returns:
        list: Các hàng trùng lặp
    """
    seen = set()
    duplicates = []

    for row in matrix:
        # Convert list sang tuple (hashable)
        row_tuple = tuple(row)

        if row_tuple in seen:
            if row not in duplicates:
                duplicates.append(row)
        else:
            seen.add(row_tuple)

    return duplicates

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

```
duplicates = find_duplicate_rows(matrix)
print(duplicates)  # [[1, 2, 3], [4, 5, 6]]
```

Bài 8: Powersets (Tất cả tập con)

```
def powerset(s):
    """
    Tạo tất cả tập con của một set

    Args:
        s (set): Set gốc

    Returns:
        list of sets: Tất cả tập con
    """
    from itertools import combinations

    s_list = list(s)
    result = [set()]  # Tập rỗng

    for i in range(1, len(s_list) + 1):
        for subset in combinations(s_list, i):
            result.append(set(subset))

    return result

# Test
s = {1, 2, 3}
subsets = powerset(s)
print(f"Số tập con: {len(subsets)}")  # 8 (2^3)
for subset in subsets:
    print(subset)
# set()
# {1}
# {2}
# {3}
# {1, 2}
# {1, 3}
# {2, 3}
# {1, 2, 3}
```

---

10. Tổng Kết và Checklist

☑ Kiến thức cần nắm vững

Cơ bản:

- [ ] Tạo set: literals `{}`, `set()`, comprehension
- [ ] Set rỗng: `set()` không phải `{}`
- [ ] Kiểm tra membership: `in`, `not in`
- [ ] Thêm: `.add()`, `.update()`
- [ ] Xóa: `.remove()`, `.discard()`, `.pop()`, `.clear()`

Set Operations:

- [ ] Union: `|`, `.union()`
- [ ] Intersection: `&`, `.intersection()`
- [ ] Difference: `-`, `.difference()`
- [ ] Symmetric Difference: `^`, `.symmetric_difference()`

Nâng cao:

- [ ] Subset/Superset: `<=`, `>=`, `<`, `>`
- [ ] Disjoint: `.isdisjoint()`
- [ ] In-place operations: `|=`, `&=`, `-=`, `^=`
- [ ] Frozenset: immutable set
- [ ] Set comprehension

Lưu ý quan trọng

1. **Set không có thứ tự** - không thể indexing
2. **Tự động loại bỏ trùng lặp** - mỗi phần tử chỉ 1 lần
3. **Elements phải immutable** - không chứa list/dict/set
4. **Fast membership test** - O(1) với `in`
5. `remove()` **raise error**, `.discard()` không lỗi
6. **Set operations** rất mạnh cho logic tập hợp
7. **Frozenset** để dùng làm key/element của set khác

Bước tiếp theo

Sau khi nắm vững Sets, bạn sẵn sàng học:

- **Tuples** - immutable sequences
- **Advanced collections**: `Counter`, `defaultdict`
- **Algorithms**: Set-based algorithms
- **Mathematical sets**: Venn diagrams, combinatorics

---

11. Câu Hỏi Thường Gặp (FAQ)

Q1: Tại sao set không có thứ tự?

A: Set implement bằng **hash table** để có O(1) lookup. Hash table không giữ thứ tự.

```
s = {3, 1, 2}
print(s)  # Thứ tự không đảm bảo, có thể {1, 2, 3} hoặc {2, 3, 1}
```

Q2: Làm sao để loại bỏ trùng lặp nhưng giữ thứ tự?

A: Dùng `dict.fromkeys()` (Python 3.7+).

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

# ❌ Set - không giữ thứ tự
unique = list(set(numbers))  # Có thể [1, 2, 3, 4]

# ☑ Dict - giữ thứ tự
unique = list(dict.fromkeys(numbers))  # [3, 1, 2, 4]
```

Q3: Khi nào dùng `.remove()` vs `.discard()`?

A:

- `.remove()`: Khi chắc chắn phần tử tồn tại
- `.discard()`: Khi không chắc, muốn an toàn

```
s = {1, 2, 3}

s.remove(2)   # OK
# s.remove(5) # ❌ KeyError

s.discard(3)  # OK
s.discard(5)  # ☑ OK, không lỗi
```

Q4: Set có thể chứa list không?

A: ❌ **Không**, vì list là mutable (unhashable).

```
# ❌ Lỗi
# s = {[1, 2], [3, 4]}  # TypeError

# ☑ Dùng tuple thay thế
s = {(1, 2), (3, 4)}  # OK

# ☑ Hoặc frozenset
s = {frozenset([1, 2]), frozenset([3, 4])}  # OK
```

Q5: Làm sao để tạo set của sets?

A: Dùng **frozenset**.

```
# ❌ Set của set - không được
# s = {{1, 2}, {3, 4}}  # TypeError

# ☑ Set của frozenset
s = {frozenset({1, 2}), frozenset({3, 4})}
print(s)  # {frozenset({1, 2}), frozenset({3, 4})}
```

---

12. Thử Thách Cuối Khóa

Challenge 1: Set Partition

Kiểm tra có thể chia set thành 2 tập con có tổng bằng nhau không.

```
"""
Input: {1, 5, 11, 5}
Output: True (có thể chia: {1, 5, 5} và {11})

Input: {1, 2, 3, 5}
Output: False

Gợi ý: Tổng phải chẵn, dùng dynamic programming
"""
```

Challenge 2: Longest Consecutive Sequence

Tìm dãy số liên tiếp dài nhất trong set.

```
"""
Input: {100, 4, 200, 1, 3, 2}
Output: 4 (dãy: 1, 2, 3, 4)

Yêu cầu: O(n) time
Gợi ý: Dùng set để check membership
"""
```

Challenge 3: Word Ladder

Tìm chuỗi chuyển đổi từ start → end, mỗi bước chỉ đổi 1 ký tự.

```
"""
Input:
```

```
start = "hit"
end = "cog"
word_set = {"hot", "dot", "dog", "lot", "log", "cog"}

Output: 5 (hit → hot → dot → dog → cog)

Gợi ý: BFS với set
"""
```

Challenge 4: Happy Number

Số happy: tổng bình phương các chữ số cuối cùng = 1.

```
"""
Input: 19
Process:
1^2 + 9^2 = 82
8^2 + 2^2 = 68
6^2 + 8^2 = 100
1^2 + 0^2 + 0^2 = 1 ☑

Output: True

Gợi ý: Dùng set để detect cycle
"""
```
