Python Programming - Dictionaries (Từ Điển)

> **Mục tiêu học tập**: Làm chủ Dictionaries - cấu trúc dữ liệu key-value mạnh mẽ để lưu trữ và truy xuất dữ liệu nhanh chóng theo khóa.

1. Giới Thiệu về Dictionaries

1.1. Dictionary là gì?

😨 Đặt vấn đề

Giả sử bạn cần lưu thông tin sinh viên: ID, tên, điểm các môn.

```
# ❌ Cách không hiệu quả - dùng nhiều list
student_ids = ["S001", "S002", "S003"]
student_names = ["An", "Bình", "Chi"]
student_math = [8.5, 7.0, 9.0]
student_physics = [7.5, 8.0, 8.5]

# Tìm điểm toán của sinh viên S002?
index = student_ids.index("S002")
math_score = student_math[index]  # Phức tạp và dễ lỗi!
```

Có cách nào tốt hơn để **ánh xạ** một giá trị (ID) sang thông tin tương ứng?

💡 Giải quyết: Dictionaries

Dictionary là cấu trúc dữ liệu lưu trữ **cặp key-value** (khóa-giá trị).

```
# ☑ Cách hiệu quả - dùng dictionary
student = {
    "id": "S001",
    "name": "An",
    "math": 8.5,
    "physics": 7.5
}

# Truy cập trực tiếp theo key
print(student["math"])  # 8.5 - Đơn giản và rõ ràng!
```

Đặc điểm của Dictionary:

- ☑ **Key-value pairs**: Mỗi key ánh xạ đến 1 value
- ☑ **Unordered** (Python < 3.7) / **Ordered** (Python 3.7+): Giữ thứ tự insertion
- ☑ **Mutable**: Có thể thay đổi, thêm, xóa

- ☑ **Keys phải unique**: Không có 2 key giống nhau
- ☑ **Keys phải immutable**: Chỉ dùng string, number, tuple làm key
- ☑ **Fast lookup**: O(1) time complexity

1.2. Khi Nào Dùng Dictionary?

Dùng Dictionary khi:

- Cần ánh xạ key → value (ví dụ: từ điển Anh-Việt)
- Cần truy cập nhanh theo identifier (ID, username, code)
- Lưu trữ dữ liệu có cấu trúc (JSON-like data)
- Đếm số lần xuất hiện (word count)

Dùng List khi:

- Chỉ cần lưu tập hợp giá trị không cần key
- Quan tâm đến thứ tự và index
- Cần slicing hoặc thao tác sequence

---

2. Tạo Dictionaries

2.1. Dictionary Literals

```
# Dictionary rỗng
empty_dict = {}

# Dictionary với string keys
person = {
    "name": "Nguyễn Minh Đạo",
    "age": 25,
    "city": "TP.HCM"
}

# Dictionary với số làm keys
scores = {
    1: "Yếu",
    2: "Trung bình",
    3: "Khá",
    4: "Giỏi"
}

# Dictionary hỗn hợp
mixed = {
    "name": "Python",
    42: "answer",
    (1, 2): "tuple key",
    True: "boolean key"
}
```

```
# ⚠ Lưu ý: Không dùng list/dict làm key (unhashable)
# invalid = {[1, 2]: "value"}  # TypeError!
```

2.2. Hàm `dict()`

```
# Từ keyword arguments
person = dict(name="Minh Đạo", age=25, city="TP.HCM")
print(person)  # {'name': 'Minh Đạo', 'age': 25, 'city': 'TP.HCM'}

# Từ list of tuples
pairs = [("a", 1), ("b", 2), ("c", 3)]
d = dict(pairs)
print(d)  # {'a': 1, 'b': 2, 'c': 3}

# Từ hai lists với zip()
keys = ["name", "age", "city"]
values = ["An", 25, "Hà Nội"]
person = dict(zip(keys, values))
print(person)  # {'name': 'An', 'age': 25, 'city': 'Hà Nội'}
```

2.3. Dictionary Comprehension

```
# Tạo dict bình phương
squares = {x: x**2 for x in range(5)}
print(squares)  # {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}

# Đảo ngược key-value
original = {"a": 1, "b": 2, "c": 3}
reversed_dict = {v: k for k, v in original.items()}
print(reversed_dict)  # {1: 'a', 2: 'b', 3: 'c'}

# Với điều kiện
numbers = [1, 2, 3, 4, 5]
even_squares = {x: x**2 for x in numbers if x % 2 == 0}
print(even_squares)  # {2: 4, 4: 16}
```

2.4. `fromkeys()` - Tạo dict với keys cho sẵn

```
# Tạo dict với giá trị mặc định
keys = ["name", "age", "city"]
person = dict.fromkeys(keys)
print(person)  # {'name': None, 'age': None, 'city': None}

# Với giá trị mặc định tùy chỉnh
person = dict.fromkeys(keys, "Unknown")
print(person)  # {'name': 'Unknown', 'age': 'Unknown', 'city': 'Unknown'}
```

```
# Ứng dụng: Khởi tạo counter
words = ["apple", "banana", "apple"]
word_count = dict.fromkeys(words, 0)
print(word_count)  # {'apple': 0, 'banana': 0}
```

---

3. Truy Cập Elements

3.1. Truy Cập với Square Brackets `[]`

```
person = {
    "name": "Minh Đạo",
    "age": 25,
    "city": "TP.HCM"
}

# Truy cập value theo key
print(person["name"])  # Minh Đạo
print(person["age"])   # 25

# ❌ KeyError nếu key không tồn tại
try:
    print(person["phone"])
except KeyError as e:
    print(f"Lỗi: Key {e} không tồn tại!")
```

3.2. Method `.get()` - An Toàn Hơn

```
person = {
    "name": "Minh Đạo",
    "age": 25,
    "city": "TP.HCM"
}

# get() trả về None nếu không tồn tại (không lỗi)
phone = person.get("phone")
print(phone)  # None

# get() với giá trị mặc định
phone = person.get("phone", "Không có")
print(phone)  # Không có

email = person.get("email", "unknown@email.com")
print(email)  # unknown@email.com

# So sánh [] vs .get()
print(person["name"])         # OK
print(person.get("name"))     # OK
```

```
# print(person["phone"])      # ❌ KeyError
print(person.get("phone"))    # ☑ None (an toàn)
```

Khi nào dùng `[]` vs `.get()`?

- `[]`: Khi **chắc chắn** key tồn tại
- `.get()`: Khi **không chắc** key có hay không

3.3. Kiểm Tra Key Tồn Tại

```
person = {"name": "An", "age": 25}

# Dùng 'in' operator
if "name" in person:
    print(f"Tên: {person['name']}")

if "phone" not in person:
    print("Không có số điện thoại")

# Ứng dụng: Validate trước khi truy cập
def get_student_score(scores, student_id):
    if student_id in scores:
        return scores[student_id]
    else:
        return "Sinh viên không tồn tại"

scores = {"S001": 8.5, "S002": 7.0}
print(get_student_score(scores, "S001"))  # 8.5
print(get_student_score(scores, "S999"))  # Sinh viên không tồn tại
```

---

4. Thêm và Sửa Elements

4.1. Thêm/Sửa với Square Brackets

```
person = {"name": "An"}

# Thêm key-value mới
person["age"] = 25
print(person)  # {'name': 'An', 'age': 25}

person["city"] = "Hà Nội"
print(person)  # {'name': 'An', 'age': 25, 'city': 'Hà Nội'}

# Sửa giá trị (key đã tồn tại)
person["age"] = 26
print(person)  # {'name': 'An', 'age': 26, 'city': 'Hà Nội'}
```

⚠ **Lưu ý:** Nếu key đã tồn tại → sửa, nếu chưa tồn tại → thêm mới.

4.2. Method `.update()` - Cập Nhật Nhiều Pairs

```
person = {"name": "An", "age": 25}

# Update từ dictionary khác
person.update({"city": "Hà Nội", "job": "Developer"})
print(person)
# {'name': 'An', 'age': 25, 'city': 'Hà Nội', 'job': 'Developer'}

# Sửa giá trị hiện tại
person.update({"age": 26, "city": "TP.HCM"})
print(person)
# {'name': 'An', 'age': 26, 'city': 'TP.HCM', 'job': 'Developer'}

# Update từ keyword arguments
person.update(name="Bình", phone="0123456789")
print(person)

# Update từ list of tuples
person.update([("email", "an@example.com")])
print(person)
```

4.3. Method `.setdefault()` - Set Nếu Chưa Có

```
person = {"name": "An"}

# Thêm key nếu chưa tồn tại
age = person.setdefault("age", 25)
print(age)     # 25
print(person)  # {'name': 'An', 'age': 25}

# Không thay đổi nếu key đã tồn tại
age = person.setdefault("age", 30)  # age vẫn là 25
print(age)     # 25
print(person)  # {'name': 'An', 'age': 25}
```

Ứng dụng: Word Count

```
text = "apple banana apple cherry banana apple"
words = text.split()

# Cách 1: Dùng setdefault
word_count = {}
for word in words:
    word_count.setdefault(word, 0)
    word_count[word] += 1
```

```
print(word_count)  # {'apple': 3, 'banana': 2, 'cherry': 1}

# Cách 2: Dùng get
word_count = {}
for word in words:
    word_count[word] = word_count.get(word, 0) + 1

print(word_count)  # {'apple': 3, 'banana': 2, 'cherry': 1}
```

---

5. Xóa Elements

5.1. `del` Statement

```
person = {"name": "An", "age": 25, "city": "Hà Nội"}

# Xóa một key
del person["age"]
print(person)  # {'name': 'An', 'city': 'Hà Nội'}

# ❌ KeyError nếu key không tồn tại
try:
    del person["phone"]
except KeyError:
    print("Key không tồn tại!")

# Xóa toàn bộ dictionary
del person
# print(person)  # NameError
```

5.2. Method `.pop()` - Xóa và Trả Về Value

```
person = {"name": "An", "age": 25, "city": "Hà Nội"}

# Pop key và lấy value
age = person.pop("age")
print(age)     # 25
print(person)  # {'name': 'An', 'city': 'Hà Nội'}

# Pop với giá trị mặc định (không lỗi nếu key không tồn tại)
phone = person.pop("phone", "Không có")
print(phone)   # Không có
print(person)  # {'name': 'An', 'city': 'Hà Nội'}

# ❌ Không có default → KeyError
try:
    person.pop("email")
```

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

5.3. Method `.popitem()` - Xóa Cặp Cuối Cùng

```
person = {"name": "An", "age": 25, "city": "Hà Nội"}

# popitem() xóa và trả về (key, value) cuối cùng (Python 3.7+)
item = person.popitem()
print(item)    # ('city', 'Hà Nội')
print(person)  # {'name': 'An', 'age': 25}

# Với dictionary rỗng → KeyError
empty = {}
try:
    empty.popitem()
except KeyError:
    print("Dictionary rỗng!")

# Ứng dụng: Stack (LIFO) với dictionary
stack = {"a": 1, "b": 2, "c": 3}
while stack:
    key, value = stack.popitem()
    print(f"{key}: {value}")
# c: 3
# b: 2
# a: 1
```

5.4. Method `.clear()` - Xóa Tất Cả

```
person = {"name": "An", "age": 25, "city": "Hà Nội"}

person.clear()
print(person)  # {} (dict rỗng, không bị xóa biến)
```

---

6. Dictionary Methods

6.1. `.keys()` - Lấy Tất Cả Keys

```
person = {"name": "An", "age": 25, "city": "Hà Nội"}

# keys() trả về dict_keys object (view object)
keys = person.keys()
print(keys)       # dict_keys(['name', 'age', 'city'])
print(type(keys)) # <class 'dict_keys'>
```

```
# Convert sang list
keys_list = list(person.keys())
print(keys_list)  # ['name', 'age', 'city']

# Duyệt qua keys
for key in person.keys():
    print(key)
# name
# age
# city

# Ngắn gọn hơn (không cần .keys())
for key in person:
    print(key)
```

6.2. `.values()` - Lấy Tất Cả Values

```
person = {"name": "An", "age": 25, "city": "Hà Nội"}

# values() trả về dict_values object
values = person.values()
print(values)  # dict_values(['An', 25, 'Hà Nội'])

# Convert sang list
values_list = list(person.values())
print(values_list)  # ['An', 25, 'Hà Nội']

# Duyệt qua values
for value in person.values():
    print(value)
# An
# 25
# Hà Nội

# Ứng dụng: Tính tổng điểm
scores = {"math": 8.5, "physics": 7.0, "chemistry": 9.0}
total = sum(scores.values())
print(f"Tổng điểm: {total}")  # 24.5

average = sum(scores.values()) / len(scores)
print(f"Điểm TB: {average:.2f}")  # 8.17
```

6.3. `.items()` - Lấy Tất Cả Cặp Key-Value

```
person = {"name": "An", "age": 25, "city": "Hà Nội"}

# items() trả về dict_items object (list of tuples)
items = person.items()
```

```
print(items)  # dict_items([('name', 'An'), ('age', 25), ('city', 'Hà Nội')])

# Convert sang list
items_list = list(person.items())
print(items_list)  # [('name', 'An'), ('age', 25), ('city', 'Hà Nội')]

# Duyệt qua items với unpacking
for key, value in person.items():
    print(f"{key}: {value}")
# name: An
# age: 25
# city: Hà Nội
```

Ứng dụng: In Bảng Điểm

```
scores = {
    "Toán": 8.5,
    "Lý": 7.0,
    "Hóa": 9.0,
    "Văn": 8.0
}

print("=== BẢNG ĐIỂM ===")
for subject, score in scores.items():
    print(f"{subject:<10} {score:>5}")

# === BẢNG ĐIỂM ===
# Toán         8.5
# Lý           7.0
# Hóa          9.0
# Văn          8.0
```

6.4. So Sánh `.keys()`, `.values()`, `.items()`

Method	Return Type	Use Case
`.keys()`	`dict_keys`	Cần chỉ các keys
`.values()`	`dict_values`	Cần chỉ các values (tính tổng, đếm)
`.items()`	`dict_items`	Cần cả key và value

```
scores = {"Toán": 8.5, "Lý": 7.0, "Hóa": 9.0}

# Chỉ cần keys
subjects = list(scores.keys())
print(subjects)  # ['Toán', 'Lý', 'Hóa']

# Chỉ cần values
```

```
all_scores = list(scores.values())
print(f"Điểm TB: {sum(all_scores) / len(all_scores)}")

# Cần cả key và value
for subject, score in scores.items():
    if score >= 8:
        print(f"{subject}: Giỏi")
```

---

7. Duyệt Qua Dictionaries

7.1. Duyệt Qua Keys (Mặc Định)

```
person = {"name": "An", "age": 25, "city": "Hà Nội"}

# Cách 1: Trực tiếp
for key in person:
    print(key)

# Cách 2: Dùng .keys() (rõ ràng hơn)
for key in person.keys():
    print(key)
```

7.2. Duyệt Qua Values

```
person = {"name": "An", "age": 25, "city": "Hà Nội"}

for value in person.values():
    print(value)
# An
# 25
# Hà Nội
```

7.3. Duyệt Qua Items (Key-Value Pairs)

```
person = {"name": "An", "age": 25, "city": "Hà Nội"}

# ☑ KHUYẾN NGHỊ - Dùng unpacking
for key, value in person.items():
    print(f"{key} → {value}")

# ❌ Không tốt - Truy cập value bằng key
for key in person:
    value = person[key]
    print(f"{key} → {value}")
```

7.4. Duyệt với `enumerate()`

```
person = {"name": "An", "age": 25, "city": "Hà Nội"}

for i, (key, value) in enumerate(person.items(), 1):
    print(f"{i}. {key}: {value}")
# 1. name: An
# 2. age: 25
# 3. city: Hà Nội
```

---

8. Dictionary Comprehension

8.1. Cú Pháp Cơ Bản

```
# Tạo dict bình phương
squares = {x: x**2 for x in range(5)}
print(squares)  # {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}

# Tạo dict từ list
fruits = ["apple", "banana", "cherry"]
fruit_lengths = {fruit: len(fruit) for fruit in fruits}
print(fruit_lengths)  # {'apple': 5, 'banana': 6, 'cherry': 6}

# Uppercase keys
names = {"an": 25, "binh": 30}
upper_names = {k.upper(): v for k, v in names.items()}
print(upper_names)  # {'AN': 25, 'BINH': 30}
```

8.2. Với Điều Kiện

```
# Lọc điểm >= 8
scores = {"Toán": 8.5, "Lý": 7.0, "Hóa": 9.0, "Văn": 7.5}
good_scores = {k: v for k, v in scores.items() if v >= 8}
print(good_scores)  # {'Toán': 8.5, 'Hóa': 9.0}

# Lọc key chẵn
numbers = {1: "odd", 2: "even", 3: "odd", 4: "even"}
evens = {k: v for k, v in numbers.items() if k % 2 == 0}
print(evens)  # {2: 'even', 4: 'even'}
```

8.3. Với If-Else

```
# Phân loại điểm
scores = {"Toán": 8.5, "Lý": 7.0, "Hóa": 9.0}
```

```
grades = {k: "Giỏi" if v >= 8 else "Khá" for k, v in scores.items()}
print(grades)  # {'Toán': 'Giỏi', 'Lý': 'Khá', 'Hóa': 'Giỏi'}

# Tăng/giảm giá
prices = {"apple": 10, "banana": 5, "cherry": 15}
new_prices = {k: v*0.9 if v > 10 else v*1.1 for k, v in prices.items()}
print(new_prices)  # {'apple': 9.0, 'banana': 5.5, 'cherry': 13.5}
```

8.4. Đảo Ngược Dictionary

```
# Swap key-value
original = {"a": 1, "b": 2, "c": 3}
reversed_dict = {v: k for k, v in original.items()}
print(reversed_dict)  # {1: 'a', 2: 'b', 3: 'c'}

# Ứng dụng: Grade to GPA
grade_to_gpa = {"A": 4.0, "B": 3.0, "C": 2.0, "D": 1.0}
gpa_to_grade = {v: k for k, v in grade_to_gpa.items()}
print(gpa_to_grade)  # {4.0: 'A', 3.0: 'B', 2.0: 'C', 1.0: 'D'}

# Tìm grade từ GPA
gpa = 3.0
grade = gpa_to_grade.get(gpa, "Unknown")
print(f"GPA {gpa} = Grade {grade}")  # GPA 3.0 = Grade B
```

---

9. Nested Dictionaries

9.1. Tạo Nested Dictionaries

```
# Dictionary chứa dictionary
students = {
    "S001": {
        "name": "An",
        "age": 20,
        "scores": {"math": 8.5, "physics": 7.0}
    },
    "S002": {
        "name": "Bình",
        "age": 21,
        "scores": {"math": 9.0, "physics": 8.5}
    }
}

# Hoặc
company = {
    "departments": {
        "IT": {
            "manager": "John",
```

```
            "employees": ["Alice", "Bob"]
        },
        "HR": {
            "manager": "Jane",
            "employees": ["Charlie", "David"]
        }
    }
}
```

9.2. Truy Cập Nested Values

```
students = {
    "S001": {
        "name": "An",
        "scores": {"math": 8.5, "physics": 7.0}
    }
}

# Truy cập từng cấp
name = students["S001"]["name"]
print(name)  # An

math_score = students["S001"]["scores"]["math"]
print(math_score)  # 8.5

# An toàn hơn với .get()
math = students.get("S001", {}).get("scores", {}).get("math", 0)
print(math)  # 8.5

# Key không tồn tại
physics = students.get("S999", {}).get("scores", {}).get("physics", 0)
print(physics)  # 0 (không lỗi)
```

9.3. Duyệt Nested Dictionaries

```
students = {
    "S001": {"name": "An", "scores": {"math": 8.5, "physics": 7.0}},
    "S002": {"name": "Bình", "scores": {"math": 9.0, "physics": 8.5}}
}

# Duyệt qua từng sinh viên
for student_id, info in students.items():
    print(f"\n{student_id}: {info['name']}")
    for subject, score in info['scores'].items():
        print(f"  {subject}: {score}")

# Output:
# S001: An
#   math: 8.5
```

```
#   physics: 7.0
#
# S002: Bình
#   math: 9.0
#   physics: 8.5
```

9.4. Ứng Dụng: Quản Lý Điểm Sinh Viên

```
# Dữ liệu từ CSV giống bài 3
scores_data = [
    ("S001", "Math", 85),
    ("S001", "Physics", 92),
    ("S001", "Chemistry", 78),
    ("S002", "Math", 88),
    ("S002", "Physics", 84),
    ("S002", "Chemistry", 90)
]

# Tạo nested dictionary
students = {}
for student_id, subject, score in scores_data:
    if student_id not in students:
        students[student_id] = {}
    students[student_id][subject] = score

print(students)
# {
#   'S001': {'Math': 85, 'Physics': 92, 'Chemistry': 78},
#   'S002': {'Math': 88, 'Physics': 84, 'Chemistry': 90}
# }

# Tìm môn có điểm cao nhất của từng sinh viên
def highest_score_subject(scores):
    """Trả về môn có điểm cao nhất"""
    return max(scores.items(), key=lambda x: x[1])

for student_id, subjects in students.items():
    subject, score = highest_score_subject(subjects)
    print(f"{student_id}: {subject} ({score})")

# Output:
# S001: Physics (92)
# S002: Chemistry (90)
```

---

10. Thao Tác Nâng Cao

10.1. Merge Dictionaries

```python
# Cách 1: update()
dict1 = {"a": 1, "b": 2}
dict2 = {"c": 3, "d": 4}

dict1.update(dict2)
print(dict1)  # {'a': 1, 'b': 2, 'c': 3, 'd': 4}

# Cách 2: Unpacking operator ** (Python 3.5+)
dict1 = {"a": 1, "b": 2}
dict2 = {"c": 3, "d": 4}
merged = {**dict1, **dict2}
print(merged)  # {'a': 1, 'b': 2, 'c': 3, 'd': 4}

# Cách 3: Merge operator | (Python 3.9+)
merged = dict1 | dict2
print(merged)  # {'a': 1, 'b': 2, 'c': 3, 'd': 4}

# Key trùng → value của dict sau ghi đè
dict1 = {"a": 1, "b": 2}
dict2 = {"b": 3, "c": 4}
merged = {**dict1, **dict2}
print(merged)  # {'a': 1, 'b': 3, 'c': 4}
```

10.2. Filter Dictionary

```python
scores = {"Toán": 8.5, "Lý": 7.0, "Hóa": 9.0, "Văn": 6.5}

# Lọc điểm >= 8
good_scores = {k: v for k, v in scores.items() if v >= 8}
print(good_scores)  # {'Toán': 8.5, 'Hóa': 9.0}

# Lọc theo key
science_subjects = {k: v for k, v in scores.items()
                    if k in ["Toán", "Lý", "Hóa"]}
print(science_subjects)  # {'Toán': 8.5, 'Lý': 7.0, 'Hóa': 9.0}
```

10.3. Sort Dictionary

```python
scores = {"Toán": 8.5, "Lý": 7.0, "Hóa": 9.0, "Văn": 6.5}

# Sắp xếp theo key
sorted_by_key = dict(sorted(scores.items()))
print(sorted_by_key)  # {'Hóa': 9.0, 'Lý': 7.0, 'Toán': 8.5, 'Văn': 6.5}

# Sắp xếp theo value (tăng dần)
sorted_by_value = dict(sorted(scores.items(), key=lambda x: x[1]))
print(sorted_by_value)  # {'Văn': 6.5, 'Lý': 7.0, 'Toán': 8.5, 'Hóa': 9.0}
```

```
# Sắp xếp theo value (giảm dần)
sorted_desc = dict(sorted(scores.items(), key=lambda x: x[1], reverse=True))
print(sorted_desc)  # {'Hóa': 9.0, 'Toán': 8.5, 'Lý': 7.0, 'Văn': 6.5}

# Top 3 điểm cao nhất
top3 = dict(sorted(scores.items(), key=lambda x: x[1], reverse=True)[:3])
print(top3)  # {'Hóa': 9.0, 'Toán': 8.5, 'Lý': 7.0}
```

10.4. Default Dictionary

```
from collections import defaultdict

# Tạo defaultdict với giá trị mặc định
word_count = defaultdict(int)  # Default = 0

text = "apple banana apple cherry banana apple"
for word in text.split():
    word_count[word] += 1  # Không cần kiểm tra key tồn tại

print(dict(word_count))  # {'apple': 3, 'banana': 2, 'cherry': 1}

# defaultdict với list
groups = defaultdict(list)
groups['fruits'].append('apple')
groups['fruits'].append('banana')
groups['vegetables'].append('carrot')

print(dict(groups))
# {'fruits': ['apple', 'banana'], 'vegetables': ['carrot']}
```

---

11. Bài Tập Thực Hành

Bài 1: Đếm tần suất ký tự

```
def count_characters(text):
    """
    Đếm số lần xuất hiện của mỗi ký tự

    Args:
        text (str): Chuỗi cần đếm

    Returns:
        dict: {char: count}
    """
    char_count = {}
    for char in text:
        char_count[char] = char_count.get(char, 0) + 1
    return char_count
```

```
# Hoặc dùng defaultdict
from collections import defaultdict

def count_characters_v2(text):
    char_count = defaultdict(int)
    for char in text:
        char_count[char] += 1
    return dict(char_count)

# Test
text = "hello world"
print(count_characters(text))
# {'h': 1, 'e': 1, 'l': 3, 'o': 2, ' ': 1, 'w': 1, 'r': 1, 'd': 1}
```

Bài 2: Tìm học sinh có điểm cao nhất (Bài 3 Assessment)

```
def highest_score_subject(student_id, scores_dict):
    """
    Tìm môn học có điểm cao nhất của sinh viên

    Args:
        student_id (str): Mã sinh viên
        scores_dict (dict): {student_id: {subject: score}}

    Returns:
        str: Tên môn học có điểm cao nhất
    """
    if student_id not in scores_dict:
        return None

    subjects = scores_dict[student_id]
    if not subjects:
        return None

    # Tìm môn có điểm cao nhất
    highest_subject = max(subjects.items(), key=lambda x: x[1])
    return highest_subject[0]

# Test với dữ liệu giống bài 3
scores = {
    "S001": {"Math": 85, "Physics": 92, "Chemistry": 78},
    "S002": {"Math": 88, "Physics": 84, "Chemistry": 90},
    "S003": {"Math": 75, "Physics": 80, "Chemistry": 83}
}

print(highest_score_subject("S001", scores))  # Physics
print(highest_score_subject("S003", scores))  # Chemistry
```

Bài 3: Group by

```
def group_by_category(items):
    """
    Nhóm items theo category

    Args:
        items (list): [(name, category), ...]

    Returns:
        dict: {category: [names]}
    """
    groups = {}
    for name, category in items:
        if category not in groups:
            groups[category] = []
        groups[category].append(name)
    return groups

# Hoặc dùng defaultdict
from collections import defaultdict

def group_by_category_v2(items):
    groups = defaultdict(list)
    for name, category in items:
        groups[category].append(name)
    return dict(groups)

# Test
items = [
    ("apple", "fruit"),
    ("carrot", "vegetable"),
    ("banana", "fruit"),
    ("broccoli", "vegetable"),
    ("orange", "fruit")
]

print(group_by_category(items))
# {
#   'fruit': ['apple', 'banana', 'orange'],
#   'vegetable': ['carrot', 'broccoli']
# }
```

Bài 4: Invert Dictionary (List Values)

```
def invert_dict(d):
    """
    Đảo ngược dictionary: value → list of keys
```

```
    Args:
        d (dict): {key: value}

    Returns:
        dict: {value: [keys]}
    """
    inverted = {}
    for key, value in d.items():
        if value not in inverted:
            inverted[value] = []
        inverted[value].append(key)
    return inverted

# Test
scores = {"An": 8, "Bình": 9, "Chi": 8, "Dũng": 7}
print(invert_dict(scores))
# {8: ['An', 'Chi'], 9: ['Bình'], 7: ['Dũng']}

# Tìm học sinh có cùng điểm
inverted = invert_dict(scores)
same_score_8 = inverted.get(8, [])
print(f"Học sinh có điểm 8: {same_score_8}")
# Học sinh có điểm 8: ['An', 'Chi']
```

Bài 5: Merge điểm từ nhiều nguồn

```
def merge_scores(*score_dicts):
    """
    Merge nhiều dictionaries điểm, tính trung bình nếu trùng key

    Args:
        *score_dicts: Các dictionaries cần merge

    Returns:
        dict: Điểm trung bình
    """
    all_scores = defaultdict(list)

    for score_dict in score_dicts:
        for student, score in score_dict.items():
            all_scores[student].append(score)

    # Tính trung bình
    result = {}
    for student, scores in all_scores.items():
        result[student] = sum(scores) / len(scores)

    return result

# Test
```

```
midterm = {"An": 8.0, "Bình": 7.5, "Chi": 9.0}
final = {"An": 8.5, "Bình": 8.0, "Chi": 9.5}
bonus = {"An": 9.0, "Chi": 10.0}

average = merge_scores(midterm, final, bonus)
print(average)
# {'An': 8.5, 'Bình': 7.75, 'Chi': 9.5}
```

Bài 6: Tìm key có value lớn nhất

```
def find_max_value_key(d):
    """
    Tìm key có value lớn nhất

    Args:
        d (dict): Dictionary số

    Returns:
        tuple: (key, value)
    """
    if not d:
        return None, None

    max_key = max(d, key=d.get)
    return max_key, d[max_key]

# Hoặc dùng items()
def find_max_value_key_v2(d):
    if not d:
        return None, None
    return max(d.items(), key=lambda x: x[1])

# Test
scores = {"Toán": 8.5, "Lý": 7.0, "Hóa": 9.0}
subject, score = find_max_value_key(scores)
print(f"Môn cao nhất: {subject} ({score})")
# Môn cao nhất: Hóa (9.0)
```

Bài 7: Dictionary từ hai lists

```
def create_dict_from_lists(keys, values, default=None):
    """
    Tạo dictionary từ list keys và list values

    Args:
        keys (list): Danh sách keys
        values (list): Danh sách values
```

```
        default: Giá trị mặc định nếu thiếu value

    Returns:
        dict: Dictionary mới
    """
    result = {}
    for i, key in enumerate(keys):
        if i < len(values):
            result[key] = values[i]
        else:
            result[key] = default
    return result

# Cách 2: Dùng zip()
def create_dict_from_lists_v2(keys, values, default=None):
    result = dict(zip(keys, values))
    # Thêm keys thiếu với default
    for key in keys:
        if key not in result:
            result[key] = default
    return result

# Test
keys = ["name", "age", "city", "phone"]
values = ["An", 25, "Hà Nội"]

d = create_dict_from_lists(keys, values, default="Unknown")
print(d)
# {'name': 'An', 'age': 25, 'city': 'Hà Nội', 'phone': 'Unknown'}
```

Bài 8: Flatten nested dictionary

```
def flatten_dict(d, parent_key='', sep='_'):
    """
    Làm phẳng nested dictionary

    Args:
        d (dict): Nested dictionary
        parent_key (str): Key cha
        sep (str): Ký tự phân cách

    Returns:
        dict: Dictionary đã flatten
    """
    items = []
    for k, v in d.items():
        new_key = f"{parent_key}{sep}{k}" if parent_key else k

        if isinstance(v, dict):
            items.extend(flatten_dict(v, new_key, sep=sep).items())
```

```
            else:
                items.append((new_key, v))

        return dict(items)

    # Test
    nested = {
        "name": "An",
        "scores": {
            "math": 8.5,
            "physics": {
                "midterm": 7.0,
                "final": 8.0
            }
        }
    }

    flat = flatten_dict(nested)
    print(flat)
    # {
    #   'name': 'An',
    #   'scores_math': 8.5,
    #   'scores_physics_midterm': 7.0,
    #   'scores_physics_final': 8.0
    # }
```

---

12. Tổng Kết và Checklist

☑ Kiến thức cần nắm vững

Cơ bản:

- ☐ Tạo dictionary: literals, `dict()`, comprehension
- ☐ Truy cập: `[]`, `.get()`
- ☐ Thêm/sửa: `[]`, `.update()`, `.setdefault()`
- ☐ Xóa: `del`, `.pop()`, `.popitem()`, `.clear()`
- ☐ Kiểm tra: `in`, `not in`

Methods:

- ☐ `.keys()`, `.values()`, `.items()`
- ☐ `.get(key, default)`
- ☐ `.update(dict)`
- ☐ `.pop(key, default)`
- ☐ `.setdefault(key, default)`

Nâng cao:

- ☐ Dictionary comprehension
- ☐ Nested dictionaries

- ☐ Merge dictionaries: `update()`, `**`, `|`
- ☐ Sort dictionary
- ☐ `defaultdict` từ collections

🎯 Lưu ý quan trọng

1. **Keys phải immutable** (string, number, tuple)
2. **Keys phải unique** (trùng → ghi đè)
3. **`[]` raise KeyError**, `.get()` trả về None/default
4. **Dictionary ordered** từ Python 3.7+
5. **`.items()` tốt nhất** khi cần cả key và value
6. **Comprehension** nhanh và Pythonic
7. **Time complexity**: Get/Set/Delete = O(1) average

📚 Bước tiếp theo

Sau khi nắm vững Dictionaries, bạn sẵn sàng học:

- **Sets** - collection không trùng lặp
- **Counter** - đếm tần suất nâng cao
- **OrderedDict** - giữ thứ tự chặt chẽ (Python < 3.7)
- **ChainMap** - kết hợp nhiều dicts

---

13. Câu Hỏi Thường Gặp (FAQ)

Q1: Khi nào dùng `[]` vs `.get()`?

A:

- `[]`: Khi **chắc chắn** key tồn tại, muốn lỗi nếu không có
- `.get()`: Khi **không chắc** key có hay không, muốn giá trị mặc định

```
config = {"debug": True, "port": 8000}

# [] - chắc chắn có
debug = config["debug"]  # OK

# get() - không chắc
timeout = config.get("timeout", 30)  # 30 (default)
```

Q2: Tại sao không dùng list làm key?

A: Vì list là **mutable** (thay đổi được), không thể hash.

```
# ❌ Lỗi
d = {[1, 2]: "value"}  # TypeError: unhashable type: 'list'
```

```
# ☑ Dùng tuple thay thế
d = {(1, 2): "value"}  # OK
```

Q3: Làm sao để check key tồn tại?

A: Dùng `in` operator.

```
person = {"name": "An", "age": 25}

# ☑ ĐÚNG
if "name" in person:
    print(person["name"])

# ❌ SAI - không dùng .keys()
if "name" in person.keys():  # Thừa
    print(person["name"])

# ☑ ĐÚNG - check value
if "An" in person.values():
    print("Found!")
```

Q4: Dictionary có giữ thứ tự không?

A:

- Python **3.7+**: ☑ Giữ thứ tự insertion
- Python **< 3.7**: ❌ Không giữ thứ tự

```
# Python 3.7+
d = {"c": 3, "a": 1, "b": 2}
print(list(d.keys()))  # ['c', 'a', 'b'] - giữ thứ tự

# Nếu cần thứ tự chắc chắn với Python cũ
from collections import OrderedDict
d = OrderedDict([("c", 3), ("a", 1), ("b", 2)])
```

Q5: Làm sao để copy dictionary?

A: Dùng `.copy()` hoặc `dict()`.

```
# ❌ Assignment không copy
dict1 = {"a": 1}
dict2 = dict1  # Reference
```

```
dict2["a"] = 100
print(dict1)  # {'a': 100} - dict1 cũng đổi!

# ☑ Shallow copy
dict1 = {"a": 1, "b": 2}
dict2 = dict1.copy()
dict2["a"] = 100
print(dict1)  # {'a': 1} - không đổi

# ⚠ Nested dict cần deep copy
import copy
dict1 = {"a": {"x": 1}}
dict2 = copy.deepcopy(dict1)
```

---

14. Thử Thách Cuối Khóa

Challenge 1: Anagram Groups

Nhóm các từ là anagram của nhau.

```
"""
Input: ["eat", "tea", "tan", "ate", "nat", "bat"]
Output: [["eat", "tea", "ate"], ["tan", "nat"], ["bat"]]

Gợi ý: Dùng sorted(word) làm key
"""
```

Challenge 2: LRU Cache

Implement cache giới hạn kích thước, xóa item ít dùng nhất.

```
"""
Yêu cầu:
- get(key): Lấy value, đưa key lên đầu (most recent)
- put(key, value): Thêm/cập nhật, xóa LRU nếu full

Example:
cache = LRUCache(2)  # capacity = 2
cache.put("a", 1)
cache.put("b", 2)
cache.get("a")       # return 1, "a" most recent
cache.put("c", 3)    # "b" bị xóa (LRU)
cache.get("b")       # return None
"""
```

Challenge 3: Word Pattern Match

Kiểm tra pattern khớp với chuỗi.

```
"""
Input: pattern = "abba", s = "dog cat cat dog"
Output: True

Input: pattern = "abba", s = "dog cat cat fish"
Output: False

Giải thích: "a" → "dog", "b" → "cat", pattern match
"""
```

Challenge 4: Top K Frequent Elements

Tìm K phần tử xuất hiện nhiều nhất.

```
"""
Input: nums = [1,1,1,2,2,3], k = 2
Output: [1, 2]

Gợi ý: Counter + sorted/heap
"""
```
