Python Programming - Built-in Functions (Hàm Tích Hợp)

> **Mục tiêu học tập**: Làm chủ các built-in functions của Python để viết code ngắn gọn, hiệu quả và Pythonic hơn.

---

1. Giới Thiệu Built-in Functions

1.1. Built-in Functions là gì?

😨 Đặt vấn đề

Giả sử bạn cần tìm số lớn nhất trong list.

```
# ❌ Cách thủ công - dài dòng
numbers = [3, 7, 2, 9, 1, 5]
max_num = numbers[0]
for num in numbers:
    if num > max_num:
        max_num = num
print(max_num)  # 9
```

Có cách nào ngắn gọn và rõ ràng hơn?

💡 Giải quyết: Built-in Functions

Built-in functions là các hàm có sẵn trong Python, không cần import.

```
# ☑ Dùng built-in function
numbers = [3, 7, 2, 9, 1, 5]
max_num = max(numbers)
print(max_num)  # 9 - Ngắn gọn, rõ ràng!
```

Lợi ích:

- ☑ Code ngắn gọn, dễ đọc
- ☑ Hiệu suất cao (implement ở C level)
- ☑ Đã test kỹ, ít bug
- ☑ Pythonic - theo chuẩn Python

1.2. Danh Sách Built-in Functions Quan Trọng

Python có **69 built-in functions**, nhưng ta sẽ tập trung vào những cái hay dùng nhất:

Basic Functions:

- `len()` - Độ dài
- `max()`, `min()` - Giá trị lớn/nhỏ nhất
- `sum()` - Tổng
- `abs()` - Giá trị tuyệt đối
- `round()` - Làm tròn

Type Conversion:

- `int()`, `float()`, `str()`, `bool()`
- `list()`, `tuple()`, `set()`, `dict()`

Iteration & Sorting:

- `sorted()` - Sắp xếp
- `enumerate()` - Index + value
- `zip()` - Kết hợp iterables
- `range()` - Sequence số

Functional Programming:

- `map()` - Áp dụng function lên iterable
- `filter()` - Lọc elements
- `all()`, `any()` - Kiểm tra điều kiện

I/O & Utility:

- `print()`, `input()`
- `type()`, `isinstance()`
- `help()`, `dir()`

---

2. Basic Functions

2.1. `len()` - Độ Dài

Trả về số lượng elements trong container.

```
# String
text = "Python"
print(len(text))  # 6

# List
numbers = [1, 2, 3, 4, 5]
print(len(numbers))  # 5

# Dictionary
person = {"name": "An", "age": 25}
print(len(person))  # 2 (số keys)
```

```
# Set
unique = {1, 2, 3}
print(len(unique))  # 3

# Range
r = range(10)
print(len(r))  # 10
```

Ứng dụng:

```
# Kiểm tra empty
if len(my_list) == 0:
    print("List rỗng")

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

# Đếm số từ
text = "Python is awesome"
word_count = len(text.split())
print(f"Số từ: {word_count}")  # 3
```

2.2. `max()` và `min()` - Giá Trị Cực Trị

`max()` - Giá trị lớn nhất

```
# Với numbers
numbers = [3, 7, 2, 9, 1, 5]
print(max(numbers))  # 9

# Với strings (so sánh alphabetically)
words = ["apple", "banana", "cherry"]
print(max(words))  # cherry

# Với nhiều arguments
print(max(5, 10, 3, 8))  # 10

# Empty iterable → ValueError
try:
    print(max([]))
except ValueError as e:
    print(f"Lỗi: {e}")  # max() arg is an empty sequence
```

`max()` với `key` parameter

```
# Tìm string dài nhất
words = ["apple", "pie", "banana", "cherry"]
longest = max(words, key=len)
print(longest)  # banana (6 ký tự)

# Tìm tuple có phần tử thứ 2 lớn nhất
students = [("An", 8), ("Bình", 9), ("Chi", 7)]
best_student = max(students, key=lambda x: x[1])
print(best_student)  # ('Bình', 9)

# Tìm dictionary có value lớn nhất
scores = {"math": 8.5, "physics": 9.0, "chemistry": 7.5}
best_subject = max(scores, key=scores.get)
print(best_subject)  # physics
```

`min()` - Giá trị nhỏ nhất

Hoạt động tương tự `max()`.

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

# Với key
words = ["apple", "pie", "banana"]
shortest = min(words, key=len)
print(shortest)  # pie
```

Ứng dụng: Tìm điểm cao/thấp nhất

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

highest = max(scores)
lowest = min(scores)
average = sum(scores) / len(scores)

print(f"Cao nhất: {highest}")
print(f"Thấp nhất: {lowest}")
print(f"Trung bình: {average:.2f}")

# Output:
# Cao nhất: 92
# Thấp nhất: 78
# Trung bình: 86.60
```

2.3. `sum()` - Tổng

Tính tổng các phần tử (phải là numbers).

```
# Cơ bản
numbers = [1, 2, 3, 4, 5]
total = sum(numbers)
print(total)  # 15

# Với start parameter (giá trị ban đầu)
total = sum(numbers, 10)  # 10 + 1 + 2 + 3 + 4 + 5
print(total)  # 25

# Tính tổng số chẵn
evens = [x for x in range(1, 11) if x % 2 == 0]
print(sum(evens))  # 30 (2 + 4 + 6 + 8 + 10)

# Hoặc dùng generator expression (tiết kiệm bộ nhớ)
total = sum(x for x in range(1, 11) if x % 2 == 0)
print(total)  # 30
```

⚠ **Lưu ý:** `sum()` chỉ dùng cho numbers, không dùng cho strings.

```
# ❌ Không dùng sum() cho strings
# strings = ["a", "b", "c"]
# result = sum(strings)  # TypeError

# ☑ Dùng join()
strings = ["a", "b", "c"]
result = "".join(strings)
print(result)  # abc
```

Ứng dụng:

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

# Đếm số phần tử thỏa điều kiện
numbers = [1, 5, 3, 8, 2, 9, 4]
count_greater_than_5 = sum(1 for x in numbers if x > 5)
print(count_greater_than_5)  # 3
```

2.4. `abs()` - Giá Trị Tuyệt Đối

```
print(abs(-5))      # 5
print(abs(5))       # 5
print(abs(-3.14))   # 3.14
```

```
# Ứng dụng: Tính khoảng cách
a = 10
b = 25
distance = abs(a - b)
print(f"Khoảng cách: {distance}")  # 15
```

2.5. round() - Làm Tròn

```
# Làm tròn đến số nguyên
print(round(3.7))    # 4
print(round(3.4))    # 3
print(round(3.5))    # 4 (banker's rounding)

# Làm tròn đến n chữ số thập phân
print(round(3.14159, 2))   # 3.14
print(round(3.14159, 3))   # 3.142

# Làm tròn đến hàng chục
print(round(127, -1))      # 130
print(round(127, -2))      # 100

# Ứng dụng: Format điểm
score = 8.756
formatted = round(score, 2)
print(f"Điểm: {formatted}")  # Điểm: 8.76
```

---

3. Type Conversion Functions

3.1. Numeric Conversions

```
# int() - Chuyển sang integer
print(int(3.7))       # 3 (cắt phần thập phân)
print(int("123"))     # 123
print(int("1010", 2)) # 10 (binary to decimal)

# float() - Chuyển sang float
print(float(5))       # 5.0
print(float("3.14"))  # 3.14

# bool() - Chuyển sang boolean
print(bool(1))        # True
print(bool(0))        # False
print(bool(""))       # False (empty string)
print(bool("text"))   # True
print(bool([]))       # False (empty list)
print(bool([1, 2]))   # True
```

3.2. String Conversions

```
# str() - Chuyển sang string
print(str(123))       # "123"
print(str(3.14))      # "3.14"
print(str([1, 2, 3])) # "[1, 2, 3]"

# chr() - ASCII code → character
print(chr(65))        # A
print(chr(97))        # a

# ord() - Character → ASCII code
print(ord('A'))       # 65
print(ord('a'))       # 97
```

3.3. Collection Conversions

```
# list() - Chuyển sang list
print(list("abc"))           # ['a', 'b', 'c']
print(list(range(5)))        # [0, 1, 2, 3, 4]
print(list({1, 2, 3}))       # [1, 2, 3]

# tuple() - Chuyển sang tuple
print(tuple([1, 2, 3]))      # (1, 2, 3)
print(tuple("abc"))          # ('a', 'b', 'c')

# set() - Chuyển sang set (loại trùng)
print(set([1, 2, 2, 3]))     # {1, 2, 3}
print(set("hello"))          # {'h', 'e', 'l', 'o'}

# dict() - Chuyển sang dictionary
pairs = [("a", 1), ("b", 2)]
print(dict(pairs))           # {'a': 1, 'b': 2}
```

---

4. Sorting & Iteration Functions

4.1. `sorted()` - Sắp Xếp

Trả về **list mới** đã sắp xếp (không thay đổi iterable gốc).

```
# Sắp xếp list
numbers = [3, 1, 4, 1, 5, 9, 2]
sorted_numbers = sorted(numbers)
print(sorted_numbers)  # [1, 1, 2, 3, 4, 5, 9]
print(numbers)         # [3, 1, 4, 1, 5, 9, 2] - không đổi

# Sắp xếp giảm dần
```

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

# Sắp xếp string
words = ["banana", "apple", "cherry"]
print(sorted(words))   # ['apple', 'banana', 'cherry']

# Sắp xếp set
unique = {3, 1, 4, 1, 5}
print(sorted(unique))  # [1, 3, 4, 5]
```

`sorted()` với `key` parameter

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

# Sắp xếp theo ký tự cuối
by_last_char = sorted(words, key=lambda x: x[-1])
print(by_last_char)  # ['apple', 'pie', 'banana', 'cherry']

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

# Sắp xếp dictionary theo value
scores = {"math": 8.5, "physics": 7.0, "chemistry": 9.0}
sorted_subjects = sorted(scores.items(), key=lambda x: x[1], reverse=True)
print(sorted_subjects)
# [('chemistry', 9.0), ('math', 8.5), ('physics', 7.0)]
```

So sánh `.sort()` vs `sorted()`

Tiêu chí	`.sort()`	`sorted()`
Return value	None	List mới
Thay đổi gốc	☑	❌
Chỉ dùng với list	☑	❌ (mọi iterable)
Hiệu suất	Nhanh hơn	Chậm hơn

```
# .sort() - in-place, chỉ list
numbers = [3, 1, 4]
numbers.sort()
print(numbers)  # [1, 3, 4]
```

```
# sorted() - tạo list mới, mọi iterable
numbers = [3, 1, 4]
result = sorted(numbers)
print(result)   # [1, 3, 4]
print(numbers)  # [3, 1, 4] - không đổi

# sorted() với set, tuple, string
print(sorted({3, 1, 4}))      # [1, 3, 4]
print(sorted((3, 1, 4)))      # [1, 3, 4]
print(sorted("python"))       # ['h', 'n', 'o', 'p', 't', 'y']
```

4.2. `enumerate()` - Index + Value

Duyệt qua iterable với cả index và value.

```
# Cơ bản
fruits = ["apple", "banana", "orange"]

for index, fruit in enumerate(fruits):
    print(f"{index}: {fruit}")

# Output:
# 0: apple
# 1: banana
# 2: orange

# Bắt đầu từ index tùy chỉnh
for index, fruit in enumerate(fruits, start=1):
    print(f"{index}. {fruit}")

# Output:
# 1. apple
# 2. banana
# 3. orange
```

So sánh với cách truyền thống:

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

# ❌ Cách cũ - không Pythonic
for i in range(len(fruits)):
    print(f"{i}: {fruits[i]}")

# ☑ Cách Pythonic
for i, fruit in enumerate(fruits):
    print(f"{i}: {fruit}")
```

Ứng dụng:

```
# Tìm index của phần tử
def find_all_indices(items, target):
    """Tìm tất cả indices của target"""
    return [i for i, x in enumerate(items) if x == target]

numbers = [1, 2, 3, 2, 4, 2, 5]
indices = find_all_indices(numbers, 2)
print(indices)  # [1, 3, 5]

# In bảng điểm có STT
students = ["An", "Bình", "Chi"]
scores = [8.5, 9.0, 7.5]

print("=== BẢNG ĐIỂM ===")
for i, (name, score) in enumerate(zip(students, scores), 1):
    print(f"{i}. {name}: {score}")

# Output:
# === BẢNG ĐIỂM ===
# 1. An: 8.5
# 2. Bình: 9.0
# 3. Chi: 7.5
```

4.3. `zip()` - Kết Hợp Iterables

Kết hợp nhiều iterables thành các tuples.

```
# Kết hợp 2 lists
names = ["An", "Bình", "Chi"]
scores = [8.5, 9.0, 7.5]

combined = zip(names, scores)
print(list(combined))
# [('An', 8.5), ('Bình', 9.0), ('Chi', 7.5)]

# Duyệt qua
for name, score in zip(names, scores):
    print(f"{name}: {score}")

# Output:
# An: 8.5
# Bình: 9.0
# Chi: 7.5
```

Kết hợp nhiều iterables

```
students = ["An", "Bình", "Chi"]
math = [8, 9, 7]
physics = [7, 8, 9]
chemistry = [9, 7, 8]

for name, m, p, c in zip(students, math, physics, chemistry):
    avg = (m + p + c) / 3
    print(f"{name}: TB = {avg:.2f}")

# Output:
# An: TB = 8.00
# Bình: TB = 8.00
# Chi: TB = 8.00
```

⚠ Zip dừng ở iterable ngắn nhất

```
a = [1, 2, 3, 4]
b = ['a', 'b', 'c']

result = list(zip(a, b))
print(result)  # [(1, 'a'), (2, 'b'), (3, 'c')] - thiếu 4

# Dùng zip_longest nếu muốn giữ hết
from itertools import zip_longest

result = list(zip_longest(a, b, fillvalue=None))
print(result)
# [(1, 'a'), (2, 'b'), (3, 'c'), (4, None)]
```

Unzip với `zip(*)`

```
# Tách các tuples thành lists riêng
pairs = [(1, 'a'), (2, 'b'), (3, 'c')]

numbers, letters = zip(*pairs)
print(numbers)  # (1, 2, 3)
print(letters)  # ('a', 'b', 'c')

# Ứng dụng: Transpose matrix
matrix = [
    [1, 2, 3],
    [4, 5, 6]
]

transposed = list(zip(*matrix))
print(transposed)
# [(1, 4), (2, 5), (3, 6)]
```

```
# Chuyển sang list of lists
transposed = [list(row) for row in zip(*matrix)]
print(transposed)
# [[1, 4], [2, 5], [3, 6]]
```

Ứng dụng: Tạo dictionary từ 2 lists

```
keys = ["name", "age", "city"]
values = ["An", 25, "Hà Nội"]

person = dict(zip(keys, values))
print(person)
# {'name': 'An', 'age': 25, 'city': 'Hà Nội'}
```

---

5. Functional Programming Functions (Optional)

5.1. `map()` - Áp Dụng Function

Áp dụng function lên từng phần tử của iterable.

```
# Bình phương các số
numbers = [1, 2, 3, 4, 5]

# Cách 1: map()
squared = map(lambda x: x**2, numbers)
print(list(squared))  # [1, 4, 9, 16, 25]

# Cách 2: List comprehension (Pythonic hơn)
squared = [x**2 for x in numbers]
print(squared)  # [1, 4, 9, 16, 25]
```

Map với nhiều iterables:

```
# Cộng 2 lists
a = [1, 2, 3]
b = [4, 5, 6]

result = map(lambda x, y: x + y, a, b)
print(list(result))  # [5, 7, 9]

# Tương đương với:
result = [x + y for x, y in zip(a, b)]
print(result)  # [5, 7, 9]
```

Map với function tự định nghĩa:

```
def fahrenheit_to_celsius(f):
    return (f - 32) * 5/9

temps_f = [32, 68, 100, 212]
temps_c = map(fahrenheit_to_celsius, temps_f)
print(list(temps_c))  # [0.0, 20.0, 37.77778, 100.0]

# Hoặc dùng comprehension
temps_c = [fahrenheit_to_celsius(f) for f in temps_f]
```

⚠ **Khuyến nghị:** Thường dùng **list comprehension** thay vì `map()` vì dễ đọc hơn.

5.2. `filter()` - Lọc Elements

Lọc các elements thỏa điều kiện.

```
# Lọc số chẵn
numbers = [1, 2, 3, 4, 5, 6, 7, 8]

# Cách 1: filter()
evens = filter(lambda x: x % 2 == 0, numbers)
print(list(evens))  # [2, 4, 6, 8]

# Cách 2: List comprehension (Pythonic hơn)
evens = [x for x in numbers if x % 2 == 0]
print(evens)  # [2, 4, 6, 8]
```

Ứng dụng:

```
# Lọc strings dài hơn 5 ký tự
words = ["apple", "pie", "banana", "cherry", "kiwi"]

long_words = filter(lambda x: len(x) > 5, words)
print(list(long_words))  # ['banana', 'cherry']

# Hoặc comprehension
long_words = [w for w in words if len(w) > 5]
print(long_words)  # ['banana', 'cherry']
```

⚠ **Khuyến nghị:** Thường dùng **list comprehension với if** thay vì `filter()`.

5.3. `all()` và `any()` - Kiểm Tra Điều Kiện

`all()` - Tất cả True?

```
# Tất cả phần tử True?
print(all([True, True, True]))   # True
print(all([True, False, True]))  # False

# Tất cả số dương?
numbers = [1, 2, 3, 4, 5]
print(all(x > 0 for x in numbers))  # True

numbers = [1, -2, 3]
print(all(x > 0 for x in numbers))  # False

# Tất cả strings không rỗng?
words = ["apple", "banana", ""]
print(all(words))  # False (có empty string)

words = ["apple", "banana", "cherry"]
print(all(words))  # True
```

`any()` **- Có ít nhất 1 True?**

```
# Có ít nhất 1 True?
print(any([False, False, True]))  # True
print(any([False, False, False])) # False

# Có số âm nào không?
numbers = [1, 2, 3, 4, 5]
print(any(x < 0 for x in numbers))  # False

numbers = [1, -2, 3]
print(any(x < 0 for x in numbers))  # True
```

Ứng dụng: Validate input

```
def validate_password(password):
    """
    Password hợp lệ nếu:
    - Ít nhất 8 ký tự
    - Có chữ hoa
    - Có chữ thường
    - Có số
    """
    checks = [
        len(password) >= 8,
        any(c.isupper() for c in password),
        any(c.islower() for c in password),
        any(c.isdigit() for c in password)
    ]
```

```
    return all(checks)

# Test
print(validate_password("Abc123"))     # False (< 8 ký tự)
print(validate_password("Abc12345"))   # True
print(validate_password("abcdefgh"))   # False (không có số, chữ hoa)
```

---

6. Utility Functions

6.1. `range()` - Sequence Số

```
# range(stop)
print(list(range(5)))           # [0, 1, 2, 3, 4]

# range(start, stop)
print(list(range(1, 6)))        # [1, 2, 3, 4, 5]

# range(start, stop, step)
print(list(range(0, 10, 2)))    # [0, 2, 4, 6, 8]
print(list(range(10, 0, -1)))   # [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

# Ứng dụng: Tạo list số
even_numbers = list(range(0, 20, 2))
print(even_numbers)  # [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

6.2. `reversed()` - Đảo Ngược

```
# Đảo ngược list
numbers = [1, 2, 3, 4, 5]
reversed_nums = list(reversed(numbers))
print(reversed_nums)  # [5, 4, 3, 2, 1]

# Đảo ngược string
text = "Python"
reversed_text = "".join(reversed(text))
print(reversed_text)  # nohtyP

# Hoặc dùng slicing (nhanh hơn)
reversed_text = text[::-1]
print(reversed_text)  # nohtyP
```

6.3. `type()` và `isinstance()`

```
# type() - Lấy kiểu dữ liệu
print(type(123))        # <class 'int'>
```

```
print(type(3.14))       # <class 'float'>
print(type("text"))     # <class 'str'>
print(type([1, 2, 3]))  # <class 'list'>

# isinstance() - Kiểm tra kiểu
x = 10
print(isinstance(x, int))     # True
print(isinstance(x, float))   # False

# Kiểm tra nhiều kiểu
y = 3.14
print(isinstance(y, (int, float)))  # True
```

---

7. Bài Tập Thực Hành

Bài 1: Statistics của list

```
def calculate_statistics(numbers):
    """
    Tính các thống kê cơ bản

    Args:
        numbers (list): Danh sách số

    Returns:
        dict: Statistics
    """
    if not numbers:
        return None

    return {
        "count": len(numbers),
        "sum": sum(numbers),
        "mean": sum(numbers) / len(numbers),
        "min": min(numbers),
        "max": max(numbers),
        "range": max(numbers) - min(numbers)
    }

# Test
numbers = [85, 92, 78, 90, 88]
stats = calculate_statistics(numbers)
print(stats)
# {
#   'count': 5,
#   'sum': 433,
#   'mean': 86.6,
#   'min': 78,
#   'max': 92,
```

```
#   'range': 14
# }
```

Bài 2: Top K elements

```python
def top_k_elements(items, k, reverse=False):
    """
    Lấy K phần tử lớn/nhỏ nhất

    Args:
        items (list): Danh sách
        k (int): Số lượng
        reverse (bool): True = lớn nhất, False = nhỏ nhất

    Returns:
        list: Top K elements
    """
    sorted_items = sorted(items, reverse=reverse)
    return sorted_items[:k]

# Test
scores = [85, 92, 78, 90, 88, 95, 82]

top_3 = top_k_elements(scores, 3, reverse=True)
print(f"Top 3: {top_3}")  # [95, 92, 90]

bottom_3 = top_k_elements(scores, 3, reverse=False)
print(f"Bottom 3: {bottom_3}")  # [78, 82, 85]
```

Bài 3: Grade students

```python
def grade_students(names, scores):
    """
    Xếp loại học sinh theo điểm

    Args:
        names (list): Tên học sinh
        scores (list): Điểm tương ứng

    Returns:
        list: [(name, score, grade), ...]
    """
    def get_grade(score):
        if score >= 9:
            return 'A'
        elif score >= 8:
            return 'B'
```

```
        elif score >= 6.5:
            return 'C'
        elif score >= 5:
            return 'D'
        else:
            return 'F'

    results = []
    for name, score in zip(names, scores):
        grade = get_grade(score)
        results.append((name, score, grade))

    # Sắp xếp theo điểm giảm dần
    return sorted(results, key=lambda x: x[1], reverse=True)

# Test
names = ["An", "Bình", "Chi", "Dũng"]
scores = [8.5, 9.0, 7.5, 6.0]

results = grade_students(names, scores)
for name, score, grade in results:
    print(f"{name}: {score} - {grade}")

# Output:
# Bình: 9.0 - A
# An: 8.5 - B
# Chi: 7.5 - C
# Dũng: 6.0 - D
```

Bài 4: Normalize scores

```
def normalize_scores(scores):
    """
    Chuẩn hóa điểm về thang 0-1

    Args:
        scores (list): Điểm gốc

    Returns:
        list: Điểm chuẩn hóa
    """
    if not scores:
        return []

    min_score = min(scores)
    max_score = max(scores)
    score_range = max_score - min_score

    if score_range == 0:
        return [0.5] * len(scores)
```

```
    normalized = [(s - min_score) / score_range for s in scores]
    return normalized

# Test
scores = [78, 85, 92, 88, 95]
normalized = normalize_scores(scores)

for original, norm in zip(scores, normalized):
    print(f"{original} → {norm:.2f}")

# Output:
# 78 → 0.00
# 85 → 0.41
# 92 → 0.82
# 88 → 0.59
# 95 → 1.00
```

Bài 5: Count word frequency

```
def word_frequency(text):
    """
    Đếm tần suất xuất hiện của từ

    Args:
        text (str): Văn bản

    Returns:
        list: [(word, count), ...] sorted by count desc
    """
    words = text.lower().split()

    # Đếm tần suất
    freq = {}
    for word in words:
        freq[word] = freq.get(word, 0) + 1

    # Sắp xếp theo count giảm dần
    sorted_freq = sorted(freq.items(), key=lambda x: x[1], reverse=True)

    return sorted_freq

# Test
text = "python is awesome python is powerful python is easy"
freq = word_frequency(text)

for word, count in freq:
    print(f"{word}: {count}")

# Output:
```

```
# python: 3
# is: 3
# awesome: 1
# powerful: 1
# easy: 1
```

Bài 6: Find pairs with sum

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

    Args:
        numbers (list): Danh sách số
        target (int): Tổng mục tiêu

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

    for num in numbers:
        complement = target - num
        if complement in seen:
            # Sắp xếp để tránh trùng (a,b) vs (b,a)
            pair = tuple(sorted([num, complement]))
            if pair not in pairs:
                pairs.append(pair)
        seen.add(num)

    return pairs

# Test
numbers = [2, 7, 11, 15, 3, 6]
target = 9

pairs = find_pairs_sum(numbers, target)
print(f"Các cặp có tổng = {target}:")
for a, b in pairs:
    print(f"{a} + {b} = {target}")

# Output:
# Các cặp có tổng = 9:
# 2 + 7 = 9
# 3 + 6 = 9
```

Bài 7: Matrix operations

```python
def matrix_sum(matrix):
    """Tính tổng tất cả phần tử trong ma trận"""
    return sum(sum(row) for row in matrix)

def row_sums(matrix):
    """Tính tổng từng hàng"""
    return [sum(row) for row in matrix]

def col_sums(matrix):
    """Tính tổng từng cột"""
    return [sum(col) for col in zip(*matrix)]

def matrix_max(matrix):
    """Tìm phần tử lớn nhất"""
    return max(max(row) for row in matrix)

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

print(f"Tổng: {matrix_sum(matrix)}")       # 45
print(f"Tổng hàng: {row_sums(matrix)}")    # [6, 15, 24]
print(f"Tổng cột: {col_sums(matrix)}")     # [12, 15, 18]
print(f"Max: {matrix_max(matrix)}")        # 9
```

Bài 8: Group by range

```python
def group_by_range(numbers, ranges):
    """
    Nhóm số theo các khoảng

    Args:
        numbers (list): Danh sách số
        ranges (list): [(min, max), ...]

    Returns:
        dict: {range: [numbers]}
    """
    groups = {range_tuple: [] for range_tuple in ranges}

    for num in numbers:
        for r_min, r_max in ranges:
            if r_min <= num <= r_max:
                groups[(r_min, r_max)].append(num)
                break
```

```
    return groups

# Test
scores = [55, 78, 92, 45, 88, 67, 95, 82, 70]
ranges = [(0, 59), (60, 79), (80, 100)]

groups = group_by_range(scores, ranges)
for range_tuple, nums in groups.items():
    print(f"{range_tuple}: {sorted(nums)}")

# Output:
# (0, 59): [45, 55]
# (60, 79): [67, 70, 78]
# (80, 100): [82, 88, 92, 95]
```

---

8. Tổng Kết và Checklist

☑ Kiến thức cần nắm vững

Basic Functions:

- ☐ `len()` - độ dài
- ☐ `max()`, `min()` - cực trị (với key)
- ☐ `sum()` - tổng
- ☐ `abs()`, `round()` - giá trị tuyệt đối, làm tròn

Type Conversion:

- ☐ `int()`, `float()`, `str()`, `bool()`
- ☐ `list()`, `tuple()`, `set()`, `dict()`
- ☐ `chr()`, `ord()` - ASCII conversion

Sorting & Iteration:

- ☐ `sorted()` - sắp xếp (với key, reverse)
- ☐ `enumerate()` - index + value
- ☐ `zip()` - kết hợp iterables
- ☐ `reversed()` - đảo ngược

Functional (Optional):

- ☐ `map()` - áp dụng function
- ☐ `filter()` - lọc elements
- ☐ `all()`, `any()` - kiểm tra điều kiện

🎯 Lưu ý quan trọng

1. `sorted()` **vs** `.sort()`: sorted() tạo mới, .sort() in-place
2. `enumerate()` **tốt hơn** `range(len())`
3. **List comprehension >** `map()`**/**`filter()` (Pythonic hơn)

4. `zip()` **dừng ở iterable ngắn nhất**
5. `max()`/`min()` **với** `key` rất mạnh
6. `all()`/`any()` **với generator** tiết kiệm bộ nhớ
7. **Type conversion** cẩn thận với data loss

Bước tiếp theo

Sau khi nắm vững Built-in Functions, bạn sẵn sàng học:

- **Lambda functions** - anonymous functions
- **Itertools module** - advanced iteration
- **Functools module** - higher-order functions
- **Custom functions** với decorators

---

9. Câu Hỏi Thường Gặp (FAQ)

Q1: Khi nào dùng list comprehension vs `map()`/`filter()`?

A: List comprehension được khuyến nghị vì dễ đọc hơn.

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

# map() - ít Pythonic
squared = list(map(lambda x: x**2, numbers))

# Comprehension - Pythonic
squared = [x**2 for x in numbers]

# filter() - ít Pythonic
evens = list(filter(lambda x: x % 2 == 0, numbers))

# Comprehension - Pythonic
evens = [x for x in numbers if x % 2 == 0]
```

Q2: `sorted()` có tốn bộ nhớ không?

A: Có, `sorted()` tạo **list mới** nên tốn bộ nhớ.

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

# sorted() - tạo list mới (tốn bộ nhớ)
sorted_nums = sorted(numbers)

# .sort() - in-place (tiết kiệm bộ nhớ)
numbers.sort()  # Thay đổi numbers trực tiếp
```

Khuyến nghị:

- Dùng `.sort()` nếu có thể thay đổi list gốc
- Dùng `sorted()` nếu cần giữ list gốc

Q3: `enumerate()` có chậm không?

A: Không, `enumerate()` rất hiệu quả (C implementation).

```
# ❌ Chậm - tạo list indices
for i in range(len(items)):
    print(i, items[i])

# ☑ Nhanh - enumerate
for i, item in enumerate(items):
    print(i, item)
```

Q4: `zip()` với iterables khác độ dài?

A: `zip()` **dừng ở iterable ngắn nhất**.

```
a = [1, 2, 3, 4]
b = ['a', 'b']

result = list(zip(a, b))
print(result)  # [(1, 'a'), (2, 'b')] - thiếu 3, 4

# Dùng zip_longest nếu muốn giữ hết
from itertools import zip_longest
result = list(zip_longest(a, b, fillvalue=None))
print(result)
# [(1, 'a'), (2, 'b'), (3, None), (4, None)]
```

Q5: `sum()` có dùng được với strings không?

A: ❌ **Không**, chỉ dùng cho numbers.

```
# ❌ Lỗi
# strings = ["a", "b", "c"]
# result = sum(strings)  # TypeError

# ☑ Dùng join()
result = "".join(["a", "b", "c"])  # "abc"

# ☑ sum() với numbers
```

```
numbers = [1, 2, 3]
total = sum(numbers)  # 6
```

---

10. Thử Thách Cuối Khóa

Challenge 1: Median Finder

Tìm median (số ở giữa) của list.

```
"""
Input: [3, 1, 4, 1, 5, 9, 2]
Output: 3 (sau khi sort: [1, 1, 2, 3, 4, 5, 9])

Nếu chẵn phần tử: trung bình 2 số giữa
Input: [1, 2, 3, 4]
Output: 2.5 ((2 + 3) / 2)
"""
```

Challenge 2: K-th Largest Element

Tìm phần tử lớn thứ k (không dùng sort).

```
"""
Input: nums = [3, 2, 1, 5, 6, 4], k = 2
Output: 5 (lớn thứ 2)

Gợi ý: Dùng heap hoặc quickselect
"""
```

Challenge 3: Group Anagrams

Nhóm các từ là anagram của nhau.

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

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

Challenge 4: Custom Sort

Sắp xếp list theo quy tắc: số chẵn trước, lẻ sau, mỗi nhóm tăng dần.

```
"""
Input: [3, 1, 4, 1, 5, 9, 2, 6, 5]
Output: [2, 4, 6, 1, 1, 3, 5, 5, 9]

Gợi ý: sorted() với key custom
"""
```
