Python Programming - Strings (Chuỗi Ký Tự)

> **Mục tiêu học tập**: Làm chủ thao tác với chuỗi ký tự - từ indexing/slicing cơ bản đến các methods nâng cao và formatting chuyên nghiệp.

1. Giới Thiệu về Strings

1.1. String là gì?

String (chuỗi ký tự) là một **sequence (dãy)** các ký tự được bao trong dấu nháy.

```
# Các cách tạo string
name = "Nguyễn Minh Đạo"
message = 'Hello, World!'
quote = """Đây là chuỗi
nhiều dòng"""

print(type(name))  # <class 'str'>
```

1.2. String là Immutable

🤯 Đặt vấn đề

```
text = "Python"
text[0] = "J"  # Muốn đổi thành "Jython"
```

Kết quả: `TypeError: 'str' object does not support item assignment`

💡 Giải thích

String trong Python là **immutable** (bất biến) - không thể thay đổi sau khi tạo.

```
# ❌ Không thể sửa trực tiếp
text = "Python"
text[0] = "J"  # Error!

# ☑ Phải tạo string mới
text = "Python"
text = "J" + text[1:]  # "Jython"
print(text)
```

Tại sao immutable?

- Tối ưu bộ nhớ (string giống nhau share cùng địa chỉ)
- Thread-safe (an toàn khi chạy đa luồng)
- Có thể dùng làm key trong dictionary

---

2. Indexing (Truy Cập Ký Tự)

2.1. Positive Indexing (Index Dương)

Mỗi ký tự có **vị trí (index)** bắt đầu từ 0.

```
text = "Python"
#       P  y  t  h  o  n
#       0  1  2  3  4  5

print(text[0])  # P (ký tự đầu tiên)
print(text[1])  # y
print(text[5])  # n (ký tự cuối)
```

Ví dụ thực tế:

```
email = "dao@example.com"

# Lấy ký tự đầu tiên của tên
first_char = email[0]
print(f"Ký tự đầu: {first_char}")  # d

# Tìm vị trí @
at_index = email.index('@')
print(f"Vị trí @: {at_index}")  # 3
```

2.2. Negative Indexing (Index Âm)

Đếm **từ cuối về đầu**, bắt đầu từ -1.

```
text = "Python"
#       P  y  t  h  o  n
#      -6 -5 -4 -3 -2 -1

print(text[-1])  # n (ký tự cuối)
print(text[-2])  # o
print(text[-6])  # P (ký tự đầu)
```

Tại sao hữu ích?

```

```

```
filename = "document.pdf"

# Lấy ký tự cuối không cần biết độ dài
last_char = filename[-1]
print(last_char)  # f

# Lấy 3 ký tự cuối (extension)
extension = filename[-3:]
print(extension)  # pdf
```

2.3. Lỗi Index Out of Range

```
text = "Python"  # Độ dài: 6 (index từ 0 đến 5)

print(text[6])   # ❌ IndexError: string index out of range
print(text[100]) # ❌ IndexError
```

Cách tránh:

```
text = "Python"

# Kiểm tra trước
index = 10
if index < len(text):
    print(text[index])
else:
    print(f"Index {index} vượt quá độ dài {len(text)}")
```

---

3. Slicing (Cắt Chuỗi)

3.1. Cú Pháp Slicing

```
string[start:stop:step]
```

- `start`: Vị trí bắt đầu (bao gồm), mặc định 0
- `stop`: Vị trí kết thúc (KHÔNG bao gồm), mặc định len(string)
- `step`: Bước nhảy, mặc định 1

3.2. Slicing Cơ Bản

```
text = "Python Programming"
#       0123456789...
```

```
# Lấy từ index 0 đến 5 (không bao gồm 6)
print(text[0:6])    # Python

# Bỏ start: lấy từ đầu
print(text[:6])     # Python

# Bỏ stop: lấy đến cuối
print(text[7:])     # Programming

# Lấy toàn bộ
print(text[:])      # Python Programming
```

Ví dụ thực tế: Xử lý họ tên

```
full_name = "Nguyễn Minh Đạo"

# Tách họ (từ đầu đến khoảng trắng đầu tiên)
space_index = full_name.index(' ')
last_name = full_name[:space_index]
print(f"Họ: {last_name}")  # Nguyễn

# Tách tên (từ khoảng trắng cuối đến hết)
first_name = full_name[full_name.rfind(' ')+1:]
print(f"Tên: {first_name}")  # Đạo
```

3.3. Slicing với Negative Index

```
text = "Python Programming"

# Lấy 6 ký tự cuối
print(text[-11:])   # Programming

# Bỏ 11 ký tự cuối
print(text[:-11])   # Python

# Lấy từ index -11 đến -5
print(text[-11:-5]) # Progra
```

3.4. Slicing với Step

```
text = "0123456789"

# Bước nhảy 2 (lấy số chẵn)
print(text[::2])    # 02468

# Bước nhảy 3
```

```
print(text[::3])    # 0369

# Đảo ngược chuỗi (step = -1)
print(text[::-1])   # 9876543210
```

Ứng dụng: Đảo ngược chuỗi

```
# Cách 1: Slicing
text = "Python"
reversed_text = text[::-1]
print(reversed_text)  # nohtyP

# Cách 2: reversed() và join()
text = "Python"
reversed_text = ''.join(reversed(text))
print(reversed_text)  # nohtyP
```

3.5. Slicing Nâng Cao

```
text = "Python Programming"

# Lấy mỗi ký tự thứ 2, từ index 1 đến 10
print(text[1:10:2])  # yhnPo

# Đảo ngược từ index 5 về 0
print(text[5:0:-1])  # nohty

# Lấy 3 ký tự đầu và 3 ký tự cuối
print(text[:3] + text[-3:])  # Pyting
```

---

4. String Methods (Phương Thức)

4.1. Case Methods (Thay đổi chữ hoa/thường)

```
text = "Python Programming"

# Chuyển thành chữ thường
print(text.lower())        # python programming

# Chuyển thành chữ HOA
print(text.upper())        # PYTHON PROGRAMMING

# Viết hoa chữ cái đầu mỗi từ
print(text.title())        # Python Programming

# Viết hoa chữ cái đầu câu
```

```
print(text.capitalize())   # Python programming

# Đảo ngược hoa/thường
print(text.swapcase())     # pYTHON pROGRAMMING
```

Ứng dụng: Chuẩn hóa input

```
# Người dùng nhập tên không đúng format
name = input("Nhập tên: ")  # người dùng nhập: "ngUYễN miNH đạO"

# Chuẩn hóa
formatted_name = name.title()
print(f"Tên chuẩn: {formatted_name}")  # Nguyễn Minh Đạo
```

4.2. Search Methods (Tìm kiếm)

`.find()` và `.rfind()`

```
text = "Python is easy. Python is powerful."

# Tìm vị trí xuất hiện ĐẦU TIÊN
pos = text.find("Python")
print(pos)  # 0

# Tìm vị trí xuất hiện CUỐI CÙNG
pos = text.rfind("Python")
print(pos)  # 16

# Không tìm thấy → trả về -1
pos = text.find("Java")
print(pos)  # -1
```

`.index()` và `.rindex()`

Giống `.find()` nhưng **raise ValueError** nếu không tìm thấy.

```
text = "Python Programming"

# Tìm thấy
pos = text.index("Python")
print(pos)  # 0

# Không tìm thấy → ValueError
try:
    pos = text.index("Java")
```

```
except ValueError:
    print("Không tìm thấy!")
```

Khi nào dùng `.find()` vs `.index()`?

- `.find()`: Khi cần kiểm tra xem có tồn tại không (-1 = không có)
- `.index()`: Khi chắc chắn tồn tại, muốn lỗi nếu không có

`.count()`

Đếm số lần xuất hiện.

```
text = "Python is easy. Python is powerful. Python is fun."

count = text.count("Python")
print(f"'Python' xuất hiện {count} lần")  # 3

# Đếm khoảng trắng
spaces = text.count(" ")
print(f"Có {spaces} khoảng trắng")  # 9

# Đếm trong khoảng nhất định
count = text.count("is", 0, 20)  # Chỉ tìm trong 20 ký tự đầu
print(count)  # 1
```

4.3. Check Methods (Kiểm tra)

`.startswith()` **và** `.endswith()`

```
filename = "document.pdf"

# Kiểm tra bắt đầu bằng...
if filename.startswith("doc"):
    print("File tài liệu")

# Kiểm tra kết thúc bằng...
if filename.endswith(".pdf"):
    print("File PDF")

# Kiểm tra nhiều khả năng
if filename.endswith((".pdf", ".docx", ".txt")):
    print("File văn bản")
```

Ứng dụng: Lọc file theo extension

```

```

```
files = ["image.jpg", "video.mp4", "song.mp3", "document.pdf"]

# Lọc file ảnh
image_files = [f for f in files if f.endswith((".jpg", ".png", ".gif"))]
print(image_files)  # ['image.jpg']

# Lọc file media
media_files = [f for f in files if f.endswith((".mp3", ".mp4", ".avi"))]
print(media_files)  # ['video.mp4', 'song.mp3']
```

`.isdigit()`, `.isalpha()`, `.isalnum()`

```
# isdigit(): Toàn số
print("123".isdigit())      # True
print("12.3".isdigit())     # False (có dấu chấm)
print("12a".isdigit())      # False

# isalpha(): Toàn chữ
print("Python".isalpha())   # True
print("Python3".isalpha())  # False (có số)

# isalnum(): Chữ hoặc số (không có ký tự đặc biệt)
print("Python3".isalnum())  # True
print("Python-3".isalnum()) # False (có dấu gạch)

# isupper(), islower()
print("PYTHON".isupper())   # True
print("python".islower())   # True

# isspace(): Toàn khoảng trắng
print("   ".isspace())      # True
print("  a ".isspace())     # False
```

Ứng dụng: Validate input

```
def validate_username(username):
    """
    Username hợp lệ nếu:
    - Chỉ chứa chữ và số
    - Độ dài 5-20 ký tự
    """
    if not username.isalnum():
        return False, "Username chỉ được chứa chữ và số"

    if not (5 <= len(username) <= 20):
        return False, "Username phải từ 5-20 ký tự"

    return True, "Username hợp lệ"
```

```
# Test
print(validate_username("minhdao123"))      # (True, 'Username hợp lệ')
print(validate_username("minh_dao"))        # (False, 'Username chỉ được chứa chữ
và số')
print(validate_username("abc"))             # (False, 'Username phải từ 5-20 ký
tự')
```

4.4. Trim Methods (Loại bỏ khoảng trắng)

```
text = "   Hello, World!   "

# Loại bỏ khoảng trắng 2 đầu
print(text.strip())   # "Hello, World!"

# Loại bỏ khoảng trắng bên trái
print(text.lstrip())  # "Hello, World!   "

# Loại bỏ khoảng trắng bên phải
print(text.rstrip())  # "   Hello, World!"

# Loại bỏ ký tự tùy chỉnh
text = "***Python***"
print(text.strip("*"))  # "Python"

# Loại bỏ nhiều ký tự
text = "...!!!Python!!!..."
print(text.strip(".!"))  # "Python"
```

Ứng dụng: Xử lý input người dùng

```
# Người dùng có thể nhập với khoảng trắng thừa
user_input = input("Nhập tên: ")  # "  Minh Đạo  "
cleaned = user_input.strip()
print(f"Tên: '{cleaned}'")  # "Minh Đạo"
```

4.5. Replace Methods

`.replace()`

```
text = "Python is easy. Python is fun."

# Thay thế tất cả
new_text = text.replace("Python", "Java")
print(new_text)  # Java is easy. Java is fun.
```

```
# Thay thế giới hạn số lần (count parameter)
new_text = text.replace("Python", "Java", 1)
print(new_text)  # Java is easy. Python is fun.

# Thay thế nhiều khoảng trắng thành 1
text = "Python    is     fun"
clean = text.replace("    ", " ").replace("   ", " ").replace("  ", " ")
print(clean)  # Python is fun

# Xóa ký tự (replace bằng empty string)
text = "Python-Programming"
no_dash = text.replace("-", "")
print(no_dash)  # PythonProgramming
```

Ứng dụng: Chuẩn hóa số điện thoại

```
def format_phone(phone):
    """
    Loại bỏ các ký tự không phải số
    Input: "+84 (90) 123-4567"
    Output: "84901234567"
    """
    phone = phone.replace(" ", "")
    phone = phone.replace("-", "")
    phone = phone.replace("(", "")
    phone = phone.replace(")", "")
    phone = phone.replace("+", "")
    return phone

phone = "+84 (90) 123-4567"
clean_phone = format_phone(phone)
print(clean_phone)  # 84901234567
```

4.6. Split và Join

`.split()`

Tách chuỗi thành **list** dựa trên delimiter.

```
text = "Python is easy"

# Split theo khoảng trắng (mặc định)
words = text.split()
print(words)  # ['Python', 'is', 'easy']

# Split theo delimiter tùy chỉnh
csv_line = "name,age,city"
parts = csv_line.split(",")
print(parts)  # ['name', 'age', 'city']
```

```
# Split giới hạn số lần
text = "one,two,three,four"
parts = text.split(",", 2)  # Chỉ split 2 lần đầu
print(parts)  # ['one', 'two', 'three,four']
```

Ứng dụng: Xử lý CSV

```
# Đọc dữ liệu CSV
csv_data = """name,age,city
Minh Đạo,25,TP.HCM
An,30,Hà Nội
Bình,22,Đà Nẵng"""

lines = csv_data.split("\n")
for line in lines[1:]:  # Bỏ header
    name, age, city = line.split(",")
    print(f"{name} - {age} tuổi - {city}")

# Output:
# Minh Đạo - 25 tuổi - TP.HCM
# An - 30 tuổi - Hà Nội
# Bình - 22 tuổi - Đà Nẵng
```

`.rsplit()`

Split từ **phải sang trái**.

```
text = "one,two,three,four"

# rsplit với maxsplit
parts = text.rsplit(",", 1)  # Split từ phải, chỉ 1 lần
print(parts)  # ['one,two,three', 'four']
```

`.join()`

Nối các phần tử trong list thành chuỗi.

```
words = ["Python", "is", "awesome"]

# Nối bằng khoảng trắng
sentence = " ".join(words)
print(sentence)  # Python is awesome

# Nối bằng dấu gạch ngang
sentence = "-".join(words)
```

```
print(sentence)  # Python-is-awesome

# Nối bằng empty string
text = "".join(words)
print(text)  # Pythonisawesome
```

Ứng dụng: Tạo path từ list

```
# Tạo đường dẫn file
path_parts = ["home", "user", "documents", "file.txt"]
path = "/".join(path_parts)
print(path)  # home/user/documents/file.txt

# Tạo URL
url_parts = ["https:", "", "example.com", "api", "users"]
url = "/".join(url_parts)
print(url)  # https://example.com/api/users
```

4.7. Padding Methods (Căn chỉnh)

`.center()`, `.ljust()`, `.rjust()`

```
text = "Python"

# Căn giữa trong 20 ký tự
print(text.center(20))      # "       Python       "
print(text.center(20, "*")) # "*******Python*******"

# Căn trái
print(text.ljust(20))       # "Python              "
print(text.ljust(20, "-"))  # "Python--------------"

# Căn phải
print(text.rjust(20))       # "              Python"
print(text.rjust(20, "0"))  # "00000000000000Python"
```

Ứng dụng: In bảng căn chỉnh

```
# Dữ liệu sinh viên
students = [
    ("An", 8.5),
    ("Bình", 7.0),
    ("Chi", 9.5)
]

# In bảng căn chỉnh
print("TÊN".ljust(15) + "ĐIỂM".rjust(10))
```

```
print("-" * 25)

for name, score in students:
    print(name.ljust(15) + str(score).rjust(10))

# Output:
# TÊN                  ĐIỂM
# -------------------------
# An                    8.5
# Bình                  7.0
# Chi                   9.5
```

`.zfill()`

Thêm số 0 vào bên trái.

```
# Định dạng số thành 5 chữ số
print("42".zfill(5))    # 00042
print("123".zfill(5))   # 00123

# Với số âm
print("-42".zfill(5))   # -0042 (dấu trừ ở đầu)

# Ứng dụng: Đánh số ID
for i in range(1, 11):
    id_number = str(i).zfill(4)
    print(f"ID: {id_number}")

# ID: 0001
# ID: 0002
# ...
# ID: 0010
```

---

5. String Concatenation (Nối Chuỗi)

5.1. Toán Tử `+`

```
first_name = "Minh"
last_name = "Đạo"

# Nối bằng +
full_name = first_name + " " + last_name
print(full_name)  # Minh Đạo

# Nối nhiều chuỗi
greeting = "Xin chào, " + first_name + " " + last_name + "!"
print(greeting)  # Xin chào, Minh Đạo!
```

⚠ **Lưu ý:** Không thể nối string với số trực tiếp.

```
age = 25
# ❌ Lỗi
message = "Tuổi: " + age  # TypeError

# ☑ Phải convert sang string
message = "Tuổi: " + str(age)
print(message)  # Tuổi: 25
```

5.2. Toán Tử * (Lặp)

```
# Lặp chuỗi
print("Python " * 3)    # Python Python Python
print("=" * 20)         # ====================
print("-" * 10)         # ----------

# Tạo separator
print("=" * 50)
print("TIÊU ĐỀ".center(50))
print("=" * 50)
```

5.3. Method `.join()`

Nối hiệu quả hơn + khi có nhiều chuỗi.

```
words = ["Python", "is", "awesome"]

# ❌ Cách không hiệu quả
result = ""
for word in words:
    result = result + word + " "
print(result)

# ☑ Cách hiệu quả
result = " ".join(words)
print(result)  # Python is awesome
```

Tại sao `.join()` hiệu quả hơn?

- String immutable → mỗi lần + tạo string mới
- `.join()` tính toán kích thước 1 lần, cấp phát bộ nhớ 1 lần

---

6. String Formatting

6.1. f-strings (Python 3.6+) - KHUYẾN NGHỊ

Cách hiện đại và dễ đọc nhất.

```
name = "Minh Đạo"
age = 25
score = 8.756

# Cơ bản
message = f"Tên: {name}, Tuổi: {age}"
print(message)  # Tên: Minh Đạo, Tuổi: 25

# Với biểu thức
print(f"Năm sau: {age + 1} tuổi")  # Năm sau: 26 tuổi

# Gọi method
print(f"Tên in hoa: {name.upper()}")  # Tên in hoa: MINH ĐẠO

# Format số
print(f"Điểm: {score:.2f}")  # Điểm: 8.76
```

Format Specifiers trong f-string:

```
number = 1234.5678

# Định dạng số thập phân
print(f"{number:.2f}")      # 1234.57 (2 chữ số thập phân)
print(f"{number:.0f}")      # 1235 (làm tròn, không thập phân)

# Thêm dấu phẩy phân cách
print(f"{number:,.2f}")     # 1,234.57

# Phần trăm
ratio = 0.856
print(f"{ratio:.1%}")       # 85.6%

# Scientific notation
print(f"{number:.2e}")      # 1.23e+03

# Alignment và width
name = "Python"
print(f"{name:<10}")        # "Python    " (trái)
print(f"{name:>10}")        # "    Python" (phải)
print(f"{name:^10}")        # "  Python  " (giữa)
print(f"{name:*^10}")       # "**Python**" (giữa với padding)
```

Ví dụ thực tế: In hóa đơn

```
items = [
    ("Áo thun", 2, 150000),
    ("Quần jean", 1, 350000),
    ("Giày", 1, 500000)
]

print("=" * 50)
print("HÓA ĐƠN MUA HÀNG".center(50))
print("=" * 50)
print(f"{'Sản phẩm':<20} {'SL':>5} {'Đơn giá':>12} {'Thành tiền':>12}")
print("-" * 50)

total = 0
for name, quantity, price in items:
    subtotal = quantity * price
    total += subtotal
    print(f"{name:<20} {quantity:>5} {price:>12,} {subtotal:>12,}")

print("-" * 50)
print(f"{'TỔNG CỘNG:':<20} {total:>29,}")
print("=" * 50)

# Output:
# ==================================================
#               HÓA ĐƠN MUA HÀNG
# ==================================================
# Sản phẩm                SL     Đơn giá   Thành tiền
# --------------------------------------------------
# Áo thun                  2      150,000      300,000
# Quần jean                1      350,000      350,000
# Giày                     1      500,000      500,000
# --------------------------------------------------
# TỔNG CỘNG:                                 1,150,000
# ==================================================
```

6.2. `.format()` Method (Python 2.7+)

```
name = "Minh Đạo"
age = 25

# Positional arguments
print("Tên: {}, Tuổi: {}".format(name, age))

# Indexed arguments
print("Tên: {0}, Tuổi: {1}".format(name, age))
print("{1} tuổi, tên {0}".format(name, age))  # Đổi thứ tự

# Named arguments
print("Tên: {n}, Tuổi: {a}".format(n=name, a=age))
```

```
# Format specifiers
score = 8.756
print("Điểm: {:.2f}".format(score))  # 8.76
```

6.3. % Formatting (Old Style)

```
name = "Minh Đạo"
age = 25
score = 8.756

# %s: string, %d: integer, %f: float
print("Tên: %s, Tuổi: %d" % (name, age))
print("Điểm: %.2f" % score)  # 8.76

# Dict-based
print("Tên: %(name)s, Tuổi: %(age)d" % {"name": name, "age": age})
```

6.4. So Sánh 3 Cách

Cách	Ưu điểm	Nhược điểm	Khi nào dùng
f-strings	Ngắn gọn, dễ đọc, nhanh	Chỉ Python 3.6+	**Mặc định** cho code mới
`.format()`	Linh hoạt, tái sử dụng template	Dài hơn f-string	Khi cần template string
`%`	Quen thuộc với C/Java	Cũ, khó đọc	Legacy code

Khuyến nghị: Dùng f-strings!

---

7. String Encoding và Decoding

7.1. Unicode trong Python

Python 3 sử dụng **Unicode** mặc định.

```
# Tiếng Việt, emoji, ký tự đặc biệt
text = "Xin chào 你好 "
print(text)  # Xin chào 你好 
print(len(text))  # 11 (đếm ký tự, không phải byte)
```

7.2. Encode và Decode

```
# String → bytes (encode)
text = "Xin chào"
encoded = text.encode('utf-8')
print(encoded)  # b'Xin ch\xc3\xa0o'
```

```
print(type(encoded))  # <class 'bytes'>

# bytes → string (decode)
decoded = encoded.decode('utf-8')
print(decoded)  # Xin chào
print(type(decoded))  # <class 'str'>
```

Ứng dụng: Đọc/ghi file với encoding

```
# Ghi file với UTF-8
with open("data.txt", "w", encoding="utf-8") as f:
    f.write("Tiếng Việt có dấu")

# Đọc file với UTF-8
with open("data.txt", "r", encoding="utf-8") as f:
    content = f.read()
    print(content)
```

---

8. Escape Characters và Raw Strings

8.1. Escape Characters

```
# \n: xuống dòng
print("Dòng 1\nDòng 2")

# \t: tab
print("Name:\tPython")

# \\: backslash
print("C:\\Users\\Desktop")

# \': nháy đơn
print('It\'s a beautiful day')

# \": nháy kép
print("He said \"Hello\"")

# \r: carriage return (ít dùng)
# \b: backspace (ít dùng)
```

8.2. Raw Strings (r"...")

Bỏ qua escape characters.

```
# Chuỗi thông thường
path = "C:\\Users\\Desktop\\file.txt"
```

```
print(path)  # C:\Users\Desktop\file.txt

# Raw string - KHÔNG xử lý escape
path = r"C:\Users\Desktop\file.txt"
print(path)  # C:\Users\Desktop\file.txt

# Hữu ích cho regex
import re
pattern = r"\d{3}-\d{3}-\d{4}"  # Regex cho số điện thoại
```

---

9. Bài Tập Thực Hành

Bài 1: Đếm từ trong câu

```
def count_words(text):
    """
    Đếm số từ trong câu

    Args:
        text (str): Câu cần đếm

    Returns:
        int: Số từ
    """
    words = text.split()
    return len(words)

# Test
sentence = "Python is an amazing programming language"
print(count_words(sentence))  # 6
```

Bài 2: Kiểm tra palindrome

```
def is_palindrome(text):
    """
    Kiểm tra chuỗi có phải palindrome không
    (đọc xuôi ngược giống nhau)

    Args:
        text (str): Chuỗi cần kiểm tra

    Returns:
        bool: True nếu là palindrome
    """
    # Chuyển thành chữ thường và bỏ khoảng trắng
    cleaned = text.lower().replace(" ", "")
    return cleaned == cleaned[::-1]
```

```
# Test
print(is_palindrome("madam"))        # True
print(is_palindrome("racecar"))      # True
print(is_palindrome("hello"))        # False
print(is_palindrome("A man a plan a canal Panama"))  # True
```

Bài 3: Tách email thành username và domain

```
def parse_email(email):
    """
    Tách email thành username và domain

    Args:
        email (str): Địa chỉ email

    Returns:
        tuple: (username, domain)
    """
    at_index = email.index('@')
    username = email[:at_index]
    domain = email[at_index+1:]

    return username, domain

# Hoặc dùng split
def parse_email_v2(email):
    parts = email.split('@')
    return parts[0], parts[1]

# Test
username, domain = parse_email("dao@example.com")
print(f"Username: {username}")  # dao
print(f"Domain: {domain}")      # example.com
```

Bài 4: Chuẩn hóa tên (Title Case)

```
def normalize_name(name):
    """
    Chuẩn hóa tên: viết hoa chữ cái đầu mỗi từ

    Args:
        name (str): Tên cần chuẩn hóa

    Returns:
        str: Tên đã chuẩn hóa
    """
```

```
    # Loại bỏ khoảng trắng thừa
    name = name.strip()

    # Chuyển thành title case
    name = name.title()

    return name

# Test
print(normalize_name("  ngUYỄN miNH đạO  "))  # Nguyễn Minh Đạo
print(normalize_name("JOHN DOE"))              # John Doe
```

Bài 5: Đếm số nguyên âm

```
def count_vowels(text):
    """
    Đếm số nguyên âm trong chuỗi

    Args:
        text (str): Chuỗi cần đếm

    Returns:
        int: Số nguyên âm
    """
    vowels = "aeiouAEIOU"
    count = 0

    for char in text:
        if char in vowels:
            count += 1

    return count

# Hoặc dùng comprehension
def count_vowels_v2(text):
    vowels = "aeiouAEIOU"
    return sum(1 for char in text if char in vowels)

# Test
print(count_vowels("Python Programming"))  # 4
print(count_vowels("Hello World"))         # 3
```

Bài 6: Xóa khoảng trắng thừa

```
def remove_extra_spaces(text):
    """
    Xóa khoảng trắng thừa (nhiều khoảng trắng → 1)
```

```
    Args:
        text (str): Chuỗi cần xử lý

    Returns:
        str: Chuỗi đã xử lý
    """
    # Cách 1: split và join
    words = text.split()
    return " ".join(words)

# Cách 2: replace nhiều lần
def remove_extra_spaces_v2(text):
    while "  " in text:
        text = text.replace("  ", " ")
    return text.strip()

# Test
text = "Python    is     awesome"
print(remove_extra_spaces(text))  # Python is awesome
```

Bài 7: Thay thế từ cấm bằng dấu sao

```
def censor_words(text, bad_words):
    """
    Thay thế từ cấm bằng ***

    Args:
        text (str): Văn bản gốc
        bad_words (list): Danh sách từ cấm

    Returns:
        str: Văn bản đã kiểm duyệt
    """
    for word in bad_words:
        # Case-insensitive replacement
        text = text.replace(word, "***")
        text = text.replace(word.capitalize(), "***")
        text = text.replace(word.upper(), "***")

    return text

# Test
text = "This is bad and very Bad and BAD"
bad_words = ["bad"]
print(censor_words(text, bad_words))
# This is *** and very *** and ***
```

Bài 8: Tìm từ dài nhất trong câu

```python
def find_longest_word(sentence):
    """
    Tìm từ dài nhất trong câu

    Args:
        sentence (str): Câu cần tìm

    Returns:
        str: Từ dài nhất
    """
    words = sentence.split()
    longest = words[0]

    for word in words:
        if len(word) > len(longest):
            longest = word

    return longest

# Hoặc dùng max()
def find_longest_word_v2(sentence):
    words = sentence.split()
    return max(words, key=len)

# Test
sentence = "Python is an amazing programming language"
print(find_longest_word(sentence))  # programming
```

---

10. Tổng Kết và Checklist

☑ Kiến thức cần nắm vững

Indexing và Slicing:

- ☐ Positive indexing (0, 1, 2, ...)
- ☐ Negative indexing (-1, -2, -3, ...)
- ☐ Slicing cơ bản `[start:stop]`
- ☐ Slicing với step `[start:stop:step]`
- ☐ Đảo ngược chuỗi `[::-1]`

String Methods:

- ☐ Case: `lower()`, `upper()`, `title()`, `capitalize()`
- ☐ Search: `find()`, `index()`, `count()`
- ☐ Check: `startswith()`, `endswith()`, `isdigit()`, `isalpha()`
- ☐ Trim: `strip()`, `lstrip()`, `rstrip()`
- ☐ Transform: `replace()`, `split()`, `join()`

- ☐ Padding: `center()`, `ljust()`, `rjust()`, `zfill()`

Concatenation & Formatting:

- ☐ Nối với `+` và `*`
- ☐ f-strings (khuyến nghị)
- ☐ `.format()` method
- ☐ `%` formatting (cũ)

🎯 Lưu ý quan trọng

1. **String là immutable** - không thể thay đổi sau khi tạo
2. **Index bắt đầu từ 0** - ký tự đầu tiên là `s[0]`
3. **Slicing không bao gồm `stop`** - `s[0:3]` lấy 0, 1, 2
4. **`.find()` trả về -1** nếu không tìm thấy
5. **`.index()` raise ValueError** nếu không tìm thấy
6. **`.split()` trả về list** - có thể dùng unpacking
7. **f-strings là best practice** cho formatting

🔜 Bước tiếp theo

Sau khi nắm vững Strings, bạn sẵn sàng học:

- **Lists** - cấu trúc dữ liệu mutable
- **Regular Expressions** - pattern matching nâng cao
- **String performance** - StringBuilder pattern
- **Text processing** - NLP cơ bản

---

11. Câu Hỏi Thường Gặp (FAQ)

Q1: Tại sao `s[0:3]` chỉ lấy 3 ký tự, không phải 4?

A: Slicing không bao gồm index `stop`. `s[0:3]` = index 0, 1, 2.

```
text = "Python"
print(text[0:3])  # Pyt (index 0, 1, 2)
```

Lý do thiết kế:

- `len(s[a:b])` = `b - a` (dễ tính toán)
- `s[:a] + s[a:]` = `s` (cắt và nối lại được)

Q2: Làm sao để check xem chuỗi có chứa substring không?

A: Dùng operator `in`.

```
text = "Python Programming"

if "Python" in text:
    print("Tìm thấy!")

if "Java" not in text:
    print("Không có Java")
```

Q3: Sự khác biệt giữa `split()` và `split(' ')`?

A:

- `split()` (không tham số): split theo **bất kỳ whitespace** nào (space, tab, newline)
- `split(' ')`: chỉ split theo **space**

```
text = "Python\tis\nawesome"

print(text.split())      # ['Python', 'is', 'awesome']
print(text.split(' '))   # ['Python\tis\nawesome'] (không split được)

text2 = "a  b"  # 2 spaces
print(text2.split())     # ['a', 'b']
print(text2.split(' '))  # ['a', '', 'b'] (có empty string)
```

Q4: Làm sao để convert string thành list các ký tự?

A: Dùng `list()` hoặc list comprehension.

```
text = "Python"

# Cách 1: list()
chars = list(text)
print(chars)  # ['P', 'y', 't', 'h', 'o', 'n']

# Cách 2: list comprehension
chars = [char for char in text]
print(chars)  # ['P', 'y', 't', 'h', 'o', 'n']
```

Q5: Cách nhanh nhất để đảo ngược chuỗi?

A: Dùng slicing `[::-1]`.

```

```

```
text = "Python"
reversed_text = text[::-1]
print(reversed_text)  # nohtyP

# So sánh với các cách khác:
# 1. reversed() + join() - chậm hơn
reversed_text = ''.join(reversed(text))

# 2. Vòng lặp - chậm nhất
reversed_text = ""
for char in text:
    reversed_text = char + reversed_text
```

Benchmark: `[::-1]` nhanh nhất vì implement ở C level.

---

12. Thử Thách Cuối Khóa

Challenge 1: Mã hóa Caesar Cipher

Viết hàm mã hóa và giải mã Caesar Cipher (dịch chuyển ký tự).

```
"""
Ví dụ:
caesar_encrypt("HELLO", 3)  → "KHOOR"
caesar_decrypt("KHOOR", 3)  → "HELLO"

Gợi ý:
- Chỉ mã hóa A-Z, a-z
- Dùng ord() và chr()
- shift = 3 nghĩa là A→D, B→E, ...
"""
```

Challenge 2: Tìm substring chung dài nhất

Tìm substring chung dài nhất giữa 2 chuỗi.

```
"""
Ví dụ:
longest_common_substring("ABABC", "BABCA")  → "BABC"
longest_common_substring("abcdef", "xyzabc")  → "abc"
"""
```

Challenge 3: Validate password phức tạp

Viết hàm kiểm tra password hợp lệ:

- Ít nhất 8 ký tự
- Có chữ hoa, chữ thường, số
- Có ít nhất 1 ký tự đặc biệt (!@#$%^&*)
- Không chứa khoảng trắng

```
"""
Ví dụ:
validate_password("Abc123!@")   → True
validate_password("abc123")     → False (thiếu chữ hoa, ký tự đặc biệt)
validate_password("Abc 123!")   → False (có khoảng trắng)
"""
```

Challenge 4: Tạo slug từ tiêu đề

Chuyển tiêu đề thành slug (URL-friendly string).

```
"""
Ví dụ:
create_slug("Python Programming 101!")  → "python-programming-101"
create_slug("  Hello, World!  ")        → "hello-world"

Quy tắc:
- Chuyển thành chữ thường
- Thay khoảng trắng bằng dấu gạch ngang
- Xóa ký tự đặc biệt
- Loại bỏ dấu gạch ngang thừa
"""
```
