



# **CHEAT SHEET**

# **KỸ THUẬT LẬP TRÌNH**

Nhóm

Phạm Trung Kiên

Trần Bình Minh

# CHƯƠNG 1: TẠI SAO NÊN HỌC LẬP TRÌNH

## Lập trình là gì

- Lập trình (Programming): viết tập hợp chỉ dẫn (program) để máy tính thực hiện.
- Là hoạt động sáng tạo, giúp tự động hóa công việc, xử lý dữ liệu, và giải quyết vấn đề.

## Máy tính & người lập trình

- Máy tính liên tục hỏi "What next?" → bạn ra lệnh cho nó bằng ngôn ngữ lập trình.
- Việc con người thấy nhàm chán (đếm, lặp, tính toán) → máy tính làm rất nhanh.

## Cấu trúc máy tính

Máy tính gồm CPU (xử lý), bộ nhớ chính (lưu tạm), bộ nhớ phụ (lưu lâu), thiết bị nhập/xuất (giao tiếp), và mạng (truyền dữ liệu). CPU thực hiện lệnh do người lập trình chỉ dẫn qua chương trình.

## Ngôn ngữ lập trình

- Ngôn ngữ bậc cao: dễ đọc, gần ngôn ngữ con người (Python, C++, Java...).
- Ngôn ngữ máy: chỉ gồm 0 và 1.
- Interpreter (thông dịch): chạy từng dòng (Python).
- Compiler (biên dịch): dịch toàn bộ rồi mới chạy (C, C++).

## Python

- Ngôn ngữ thông dịch, dễ học, mạnh mẽ, linh hoạt.
- Chạy trực tiếp trong Python Shell:

```
>>> print("Hello, world")
>>> x = 5
>>> x + 3
8
>>> quit()
```

## Biến & dữ liệu

Variable: vùng nhớ lưu giá trị tạm thời.

```
x = 6
y = x * 7
print(y) # 42
```

Tên biến là nhãn (label) giúp truy cập dữ liệu

## Các lỗi thường gặp trong Python:

- Syntax Error: Viết sai cú pháp, ví dụ `primt("Hi")`
- Runtime Error: Lỗi xuất hiện khi chương trình đang chạy (ví dụ chia cho 0).
- Semantic Error: Cú pháp đúng nhưng kết quả sai ý, ví dụ tính sai công thức.

## Quy trình cơ bản

Input → Processing → Output

↓

Lưu tạm / lâu dài

## Vì sao nên học lập trình

- Tự động hóa công việc.
- Phát triển tư duy logic & sáng tạo.
- Tạo ra công cụ, ứng dụng phục vụ học tập & đời sống.

# Chương 2: Biến, Biểu thức và Câu lệnh trong Python

## Giá trị & Kiểu dữ liệu (Values & Types)

- Giá trị (Value): Các đối tượng cơ bản mà chương trình xử lý.
- Kiểu (Type) Phân loại giá trị.
- Integer (int) Số nguyên (không có phần thập phân).
- Float Số thực (có phần thập phân), dùng định dạng floating point.
- String (str) Chuỗi ký tự, được đặt trong dấu ngoặc kép ("..." hoặc '...').

## Biến (Variables)

- Biến (Variable) Một tên dùng để tham chiếu đến một giá trị.
- Lệnh Gán (Assignment) Tạo biến mới và gán giá trị cho chúng. Kiểu của biến là kiểu của giá trị nó tham chiếu.
- Quy tắc đặt tên\* Bắt đầu bằng chữ cái hoặc dấu gạch dưới (\_). \* Không được bắt đầu bằng số. \* Có thể chứa chữ cái, số, và \_ \* Nên đặt tên có ý nghĩa (gợi nhớ - mnemonic).
- Từ khóa (Keywords) Các từ được Python dành riêng để nhận dạng cấu trúc chương trình (ví dụ: if, for, class, True, def). Không được dùng làm tên biến.

## Biểu thức & Câu lệnh (Expressions & Statements)

- Câu lệnh (Statement) Một đơn vị code mà trình thông dịch Python có thể thực thi.
- Biểu thức (Expression) Sự kết hợp của giá trị, biến và toán tử, cho ra một giá trị kết quả duy nhất.

## Toán tử & Thứ tự ưu tiên (Operators & Precedence)

Toán tử:

+: Cộng hai toán hạng.

-: Trừ hai toán hạng.

\*: Nhân hai toán hạng.

/: Chia (luôn trả về float).

//: Chia lấy phần nguyên (truncated/floored division).

\*\*: Lũy thừa.

%: Lấy phần dư khi chia.

Thứ tự Ưu tiên (PEMDAS):

1. Parentheses: Dấu ngoặc đơn ()

2. Exponentiation: Lũy thừa \*\*

3. Multiplication & Division: Nhân \*, Chia /, Chia sàn //, Modulus % (thực hiện từ trái sang phải)

4. Addition & Subtraction: Cộng +, Trừ - (thực hiện từ trái sang phải)

## Thao tác Chuỗi (String Operations)

- Nối chuỗi (Concatenation) + Nối hai chuỗi lại với nhau.  
vd: '100' + '150' -> '100150'
- Lưu ý: Nếu toán hạng là số, + là phép cộng. Nếu là chuỗi, + là phép nối.

## Bình luận (Comments)

- #: Bắt đầu một bình luận. Mọi thứ từ # đến cuối dòng bị trình thông dịch bỏ qua
- Cộng dụng: Giải thích tại sao code làm điều đó, đặc biệt là các logic không rõ ràng, chứ không chỉ là nó làm gì.

## Nhập liệu từ người dùng (Input)

- Nhập liệu input([prompt]) Tạm dừng chương trình, chờ người dùng nhập và nhấn Enter. Luôn trả về giá trị String (str).
- Chuyển kiểu int(value) float(value) Chuyển đổi giá trị string thành integer hoặc float.
- Ký tự đặc biệt \n Ký tự xuống dòng (Newline), thường dùng trong prompt của input().

## Gỡ Lỗi (Debugging)

### 1. Lỗi Cú Pháp (Syntax Errors)

- Nguyên nhân: Vi phạm quy tắc ngôn ngữ Python (ví dụ: dùng khoảng trắng, từ khóa, ký tự bất hợp pháp trong tên biến).
- Đặc điểm: Chương trình không chạy được, báo lỗi như `SyntaxError: invalid syntax`.

### 2. Lỗi Thời Gian Chạy (Runtime Errors)

- Nguyên nhân: Cố gắng sử dụng biến chưa được gán giá trị (lỗi "use before def") hoặc gõ sai chính tả tên biến (ví dụ: `NameError`).
- Đặc điểm: Chương trình chạy, nhưng dừng lại đột ngột khi gặp lỗi.

### 3. Lỗi Ngữ Nghĩa (Semantic Errors)

- Nguyên nhân: Lỗi logic hoặc hiểu sai thứ tự ưu tiên toán tử (ví dụ: thiếu dấu ngoặc đơn).
- Đặc điểm: Chương trình chạy hoàn hảo, nhưng cho ra kết quả sai mà không báo lỗi. (Đây là lỗi khó tìm nhất!)

## Glossary

Danh Mục	Thuật Ngữ	Định Nghĩa
Cấu Trúc Mã Lệnh	assignment (Gán)	Một câu lệnh dùng để <b>gán một giá trị</b> vào một biến.
	comment (Chú thích)	Thông tin trong chương trình dành cho lập trình viên, <b>không ảnh hưởng</b> đến việc thực thi mã.
	keyword (Từ khóa)	Từ dành riêng của Python, <b>không được</b> dùng làm tên biến (ví dụ: if, def).
	mnemonic (Gợi nhớ)	Một công cụ hỗ trợ trí nhớ; đặt tên biến để <b>nhớ</b> .
Toán Tử và Biểu Thức	expression (Biểu thức)	Tổ hợp biến, toán tử, giá trị, đại diện cho <b>một kết quả đơn lẻ</b> .
	evaluate (Đánh giá)	Đơn giản hóa một biểu thức bằng cách thực hiện các phép toán theo thứ tự ưu tiên.
	operator (Toán tử)	Một ký hiệu đặc biệt đại diện cho một phép tính đơn giản.
	operand (Toán hạng)	Một trong các giá trị mà toán tử tác động lên.
	rules of precedence (Quy tắc ưu tiên)	Tập hợp các quy tắc chi phối <b>thứ tự</b> đánh giá các phép toán.
	modulus operator (%)	Toán tử cho ra <b>phần dư</b> của phép chia số nguyên.
	concatenate (Nối chuỗi)	Nối hai toán hạng (chuỗi) <b>liên kề nhau</b> .
Các Loại Dữ Liệu	integer (Số nguyên)	Kiểu dữ liệu đại diện cho <b>số nguyên</b> (không có phần thập phân).
	floating point (Số thực)	Kiểu dữ liệu đại diện cho các số <b>có phần thập phân</b> .

Danh Mục	Thuật Ngữ	Định Nghĩa
Cấu Trúc Mã Lệnh	<b>assignment</b> (Gán)	Một câu lệnh dùng để <b>gán một giá trị</b> vào một biến.
	<b>comment</b> (Chú thích)	Thông tin trong chương trình dành cho lập trình viên, <b>không ảnh hưởng</b> đến việc thực thi mã.
	<b>keyword</b> (Từ khóa)	Từ dành riêng của Python, <b>không được</b> dùng làm tên biến (ví dụ: if, def).
	<b>mnemonic</b> (Gợi nhớ)	Một công cụ hỗ trợ trí nhớ; đặt tên biến <b>để nhớ</b> .
Toán Tử và Biểu Thức	<b>expression</b> (Biểu thức)	Tổ hợp biến, toán tử, giá trị, đại diện cho <b>một kết quả đơn lẻ</b> .
	<b>evaluate</b> (Đánh giá)	Đơn giản hóa một biểu thức bằng cách thực hiện các phép toán theo thứ tự ưu tiên.
	<b>operator</b> (Toán tử)	Một ký hiệu đặc biệt đại diện cho một phép tính đơn giản.
	<b>operand</b> (Toán hạng)	Một trong các giá trị mà toán tử tác động lên.
	<b>rules of precedence</b> (Quy tắc ưu tiên)	Tập hợp các quy tắc chi phối <b>thứ tự</b> đánh giá các phép toán.
	<b>modulus operator</b> (%)	Toán tử cho ra <b>phần dư</b> của phép chia số nguyên.
	<b>concatenate</b> (Nối chuỗi)	Nối hai toán hạng (chuỗi) <b>liên kề nhau</b> .
Các Loại Dữ Liệu	<b>integer</b> (Số nguyên)	Kiểu dữ liệu đại diện cho <b>số nguyên</b> (không có phần thập phân).
	<b>floating point</b> (Số thực)	Kiểu dữ liệu đại diện cho các số <b>có phần thập phân</b> .

# Chương 3: THỰC THI CÓ ĐIỀU KIỆN (CONDITIONAL EXECUTION)

## 1. Biểu Thức Boolean & So Sánh

Toán tử	Ý nghĩa	Ví dụ	Kết quả
=	Bằng	5 == 5	TRUE
!=	Không bằng	5 != 6	TRUE
>	Lớn hơn	x > y	False/True
<	Nhỏ hơn	x < y	False/True
>=	Lớn hơn hoặc bằng	x >= y	False/True
<=	Nhỏ hơn hoặc bằng	x <= y	False/True
is	Cùng là một đối tượng	x is y	False/True
is not	Không cùng là một đối tượng	x is not y	False/True

## 2. Toán Tử Logic (Logical Operators)

Toán tử	Ý nghĩa	Ví dụ	Điều kiện Trả về True
and	Và	x > 0 and x < 10	Cả hai điều kiện đều đúng.
or	Hoặc	n%2 == 0 or n%3 == 0	Ít nhất một điều kiện đúng.
not	Phủ định	not (x > y)	Biểu thức x > y là sai (tức là x <= y)

## 3. Thực Thi Có Điều Kiện (Conditional Execution)

- A. If (Thực thi đơn giản)  
Thực hiện khối lệnh chỉ khi điều kiện là True.  
if x > 0:  
    print("x is positive")
- B. If-Else (Thực thi thay thế)  
Có hai nhánh (branches), chỉ một trong hai được thực thi.  
if x % 2 == 0:  
    print('x is even')  
else:  
    print('x is odd')
- C. Chained Conditionals (Điều kiện nối chuỗi)  
Sử dụng elif (viết tắt của "else if") để kiểm tra nhiều khả năng. Chỉ nhánh đầu tiên đúng được thực thi.  
if x < y:  
    print('x is less than y')  
elif x > y:  
    print('x is greater than y')  
else:  
    print('x and y are equal')
- D. Nested Conditionals (Điều kiện lồng nhau)  
if 0 < x:  
    if x < 10:  
        print('single-digit')  
if 0 < x and x < 10:  
    print('single-digit')

## 4. Xử Lý Ngoại Lệ: try và except

Cấu trúc try / except là một "chính sách bảo hiểm" cho các khối lệnh có thể gây ra lỗi (ngoại lệ - exception), giúp chương trình không bị dừng đột ngột (crash).

- try block: Chứa các lệnh có thể gây lỗi.
- except block: Chứa các lệnh được thực thi chỉ khi một lỗi xảy ra trong khối try

```
inp = input('Enter number:')  
try:  
    fahr = float(inp) # Lệnh này có thể gây lỗi  
ValueError  
    cel = (fahr - 32.0) * 5.0 / 9.0  
    print(cel)  
except:  
    print('Please enter a number') # Khối này chạy nếu ValueError xảy ra
```

## 5. Gỡ Lỗi (Debugging)

- Traceback: Danh sách các hàm đang thực thi khi lỗi xảy ra. Thông tin quan trọng nhất là loại lỗi và vị trí lỗi xảy ra.
- Lỗi Thụt lề (IndentationError): Lỗi do khoảng trắng/tab không đúng. Thông báo lỗi thường chỉ vào dòng sau vị trí lỗi thực tế.

Lỗi thụt lề phổ biến:

```
x = 5  
y = 6 # ✗ IndentationError
```

## 6. Thuật ngữ (Glossary)

- body  
→ Thân lệnh: tập hợp các câu lệnh nằm bên trong một câu lệnh phức (compound statement), ví dụ như phần thụt lề trong if, for, while, def.
- boolean expression  
→ Biểu thức Boolean: biểu thức cho ra giá trị True hoặc False.
- branch  
→ Nhánh: một trong các đường thực thi khác nhau trong câu lệnh điều kiện (như nhánh if, elif, else).
- chained conditional  
→ Câu điều kiện chuỗi: câu điều kiện gồm nhiều nhánh nối tiếp nhau (if → elif → else).
- comparison operator  
→ Toán tử so sánh: các toán tử dùng để so sánh hai giá trị: ==, !=, >, <, >=, <=.
- conditional statement  
→ Câu lệnh điều kiện: câu lệnh điều khiển luồng thực thi dựa trên một điều kiện (thường là câu lệnh if).

# CHƯƠNG 4:FUNCTIONS (HÀM)

## Function Calls (Gọi hàm)

Định Nghĩa

- Hàm là tập hợp câu lệnh có tên, thực hiện một tác vụ.
- Khi gọi hàm, ta truyền đối số (argument) và nhận kết quả (return value).

Ví dụ

```
1 A=12
2 print(type(A))
3
```

## Built-in Functions (Hàm có sẵn)

Định nghĩa

max(), min() → tìm giá trị lớn nhất, nhỏ nhất.

len() → đếm số phần tử hoặc độ dài chuỗi.

print()

input()

float()

## Type Conversion Functions (Chuyển kiểu dữ liệu)

int(x) → đổi x sang số nguyên.

float(x) → đổi x sang số thực.

str(x) → đổi x sang chuỗi.

```
A=input(float(32))
print(A)
```

## Random Numbers (Số ngẫu nhiên)

Các hàm phổ biến:

-random.random() → số thực từ 0.0 đến <1.0

-random.randint(a, b) → số nguyên trong [a, b]

-random.choice(list) → chọn ngẫu nhiên phần tử trong danh sách

```
import random
print(random.randint(1,10))
```

## Math Functions (Hàm toán học)

Dùng module math

```
import math
print(math.sin(math.pi/2))
```

math.pi là giá trị  $\pi$  (3.14159...)

## Adding New Functions (Tạo hàm mới)

Hàm phải được định nghĩa trước khi gọi.

Trình tự thực thi:

1.Python đọc def và lưu hàm.

2.Khi gọi tên hàm → các câu lệnh bên trong chạy.

```
def print_lyrics():
    print("Hi:")
print_lyrics()
```

## Flow of Execution (Luồng thực thi)

Python chạy từ trên xuống dưới.  
Khi gặp lời gọi hàm → tạm dừng, chạy nội dung hàm, sau đó quay lại vị trí gọi.

## Parameters and Arguments (Tham số & Đối số)

Parameter: biến trong định nghĩa hàm.  
Argument: giá trị được truyền khi gọi hàm.

```
def greet(name):  
    print("Hello", name)  
  
greet("Alice")
```

## Fruitful and Void Functions

Fruitful function: có return, trả về giá trị.

```
def add(a,b):  
    return a+b
```

Void function: không return, chỉ thực hiện hành động.

```
def greet():  
    print("Hi")
```

## Why Functions? (Tại sao dùng hàm)

Giúp tái sử dụng mã, chia nhỏ chương trình, dễ đọc – dễ sửa.  
Giúp tránh trùng lặp và giảm lỗi.

## Debugging (Gỡ lỗi)

Kiểm tra xem:

- Hàm đã định nghĩa trước khi gọi chưa?
- Có trả về giá trị mong muốn không?
- Có lỗi về thụt lề hay không?

## Glossary (Từ vựng quan trọng)

- Algorithm – thuật toán.
- Argument – giá trị truyền vào hàm.
- Body – phần thân của hàm.
- Flow of execution – trình tự chạy chương trình.
- Fruitful function – hàm có giá trị trả về.
- Void function – hàm không trả về giá trị.
- Return value – kết quả của hàm.

# CHƯƠNG 5: LOOP AND ITERATION (VÒNG LẶP)

## Updating Variables (Cập nhật biến)

```
x = x + 1
```

## The while Statement (Câu lệnh while)

```
n = 5
while n > 0:
    print(n)
    n = n - 1
print('End')
```

Kiểm tra điều kiện  
(True/False)  
Nếu True → chạy thân  
vòng lặp  
Nếu False → thoát vòng  
lặp

## Infinite Loops (Vòng lặp vô hạn)

```
while True:
    print("Hello")
```

Nếu điều kiện luôn đúng (while True:),  
vòng lặp sẽ chạy mãi

## Using break to Exit a Loop

```
while True:
    line = input(' ')
    if line == 'done':
        break
    print(line)
print('Done!')
```

break dùng để thoát vòng lặp ngay lập  
tức, dù điều kiện còn đúng.

## Finishing Iterations with continue

continue bỏ qua phần còn lại của vòng lặp  
hiện tại → chuyển sang vòng tiếp theo.

```
while True:
    line = input('> ')
    if line[0] == '#':
        continue
    if line == 'done':
        break
    print(line)
print('Done!')
```

## Definite Loops using for (Vòng lặp for)

Dùng khi biết trước số lần lặp  
hoặc có danh sách dữ liệu.

```
for i in [5, 4, 3, 2, 1]:
    print(i)
print('Blastoff!')
```

## Loop Patterns (Mẫu lặp thường dùng)

```
count = 0
sum = 0
for value in [3, 41, 12, 9, 74, 15]:
    count = count + 1
    sum = sum + value
print('Count:', count, 'Sum:', sum)
```

count đếm số phần tử  
sum cộng dồn giá trị

## Glossary (Từ vựng quan trọng)

Iteration	Quá trình lặp lại các câu lệnh
Loop	Cấu trúc cho phép lặp
While loop	Lặp khi điều kiện còn đúng
For loop	Lặp qua các phần tử trong danh sách
Break	Dừng vòng lặp
Continue	Bỏ qua phần còn lại của vòng hiện tại
Infinite loop	Vòng lặp vô hạn

# CHƯƠNG 6: STRINGS (CHUỖI)

## A String is a Sequence

- Chuỗi (string) là một dãy ký tự trong ngoặc đơn hoặc ngoặc kép.
- Các ký tự trong chuỗi có vị trí (index) bắt đầu từ 0:
- Nếu truy cập chỉ số ngoài phạm vi → lỗi IndexError.

## Getting the Length of a String – len()

Hàm len() trả về độ dài chuỗi.

```
A=len('banana')  
print(A)
```

## Traversal Through a String with a Loop

Duyệt từng ký tự trong chuỗi bằng while hoặc for:

## String Slices

Dùng cú pháp [start:end] để trích xuất một phần của chuỗi. Chỉ mục end không được bao gồm.

Hành động	Cú pháp	Ví dụ	Kết quả
Từ đầu đến n	chuoi[:n]	"Python"[:2]	"Py"
Từ n đến cuối	chuoi[n:]	"Python"[2:]	"thon"
Giữa	chuoi[start:end]	"Python"[1:4]	"yth"
Tạo bản sao	chuoi[:]	"Python"[:]	"Python"

## Strings are Immutable (Không thể thay đổi)

Chuỗi không thể bị thay đổi sau khi được tạo. Mọi thao tác dường như chỉnh sửa chuỗi thực chất là tạo ra một chuỗi mới.

VD: greeting[0] = 'J' --> TypeError

## Looping and Counting (Đếm ký tự)

- Dùng vòng lặp for để duyệt từng ký tự trong chuỗi.
- Dùng biến đếm (counter) khởi tạo bằng 0.
- Mỗi lần gặp ký tự cần tìm → tăng biến đếm: count = count + 1
- Sau vòng lặp, count chứa tổng số lần xuất hiện.

```
fruit = 'banana'  
for char in fruit:  
    print(char)
```

## The in Operator

- in là toán tử boolean.
- Trả về True nếu chuỗi A nằm trong chuỗi B (substring).
- Ngược lại trả về False.

```
>>> 'a' in 'banana'
True
>>> 'seed' in 'banana'
False
```

## String Comparison

- == : kiểm tra hai chuỗi có giống nhau không
- ```
if word == 'banana':
    print("All right, bananas.")
```
- < , > : so sánh theo thứ tự từ điển (alphabetical order)
- ```
if word < 'banana':
    print("comes before banana")
elif word > 'banana':
    print("comes after banana")
```

## String Methods (Các phương thức chuỗi)

- String là object → có dữ liệu + các phương thức (methods).
- Dùng type(obj) để xem kiểu, dir(obj) để xem toàn bộ methods.
- Gọi method bằng dot notation:
- Các method hay dùng:

upper() / lower() → đổi chữ hoa/thường.  
find(sub) → tìm vị trí chuỗi con.  
strip() → bỏ khoảng trắng đầu/cuối.  
startswith() / endswith() → trả về True/False.  
replace(old, new) → thay thế chuỗi.  
split() → tách chuỗi.  
join(list) → nối list thành chuỗi.

## Parsing Strings (Tách chuỗi)

Quy trình chung

1. Dùng find() để tìm vị trí ký tự/mẫu cần tìm.
2. Dùng find() lần nữa để tìm vị trí kết thúc.
3. Dùng slicing để lấy phần chuỗi mong muốn.

## Format Operator %

Dùng để chèn giá trị vào chuỗi theo định dạng.

Cú pháp : "format\_string" % value

Format phổ biến:

- %d → số nguyên (decimal)
- %f → số thực
- %s → chuỗi (string)

## Glossary (Từ vựng)

- counter → biến đếm (thường bắt đầu từ 0, tăng dần).
- empty string → chuỗi rỗng "", độ dài 0.
- format operator (%) → chèn giá trị vào chuỗi theo định dạng.
- format sequence → ký hiệu định dạng trong chuỗi, ví dụ: %d, %s, %f.
- format string → chuỗi chứa format sequence để dùng với %.
- flag → biến boolean dùng để báo hiệu trạng thái Đúng/Sai.
- invocation → lời gọi phương thức (method call).
- immutable → không thể thay đổi giá trị bên trong (string là immutable).
- index → vị trí của phần tử trong sequence (bắt đầu từ 0).
- item → phần tử trong một sequence.
- method → hàm gắn với object, gọi bằng dấu chấm.
- object → giá trị mà biến trỏ tới (value).
- search → duyệt chuỗi cho đến khi tìm thấy mục tiêu.
- sequence → tập giá trị có thứ tự, truy cập bằng index (string, list, tuple).
- slice → cắt một phần chuỗi bằng chỉ số [start:end].
- traverse → duyệt từng phần tử trong sequence.

# Chương 7: Files

## Persistence

### 1. Khái Niệm Cơ Bản

- Bộ nhớ Chính (Main Memory): Là nơi phần mềm hoạt động và chạy (CPU và Main Memory). Toàn bộ quá trình "suy nghĩ" của chương trình xảy ra ở đây.
- Dữ liệu "Tạm thời" (Transient): Dữ liệu được lưu trữ trong CPU hoặc Main Memory (RAM) sẽ bị xóa ngay khi máy tính bị tắt nguồn. Các chương trình đã học cho đến nay đều là các chương trình tạm thời.

### 2. Bộ Nhớ Thứ Cấp (Secondary Memory)

- Định nghĩa: Bộ nhớ thứ cấp (hay còn gọi là Tập/Files) là giải pháp cho vấn đề mất dữ liệu khi tắt nguồn. \* Đặc tính: Dữ liệu không bị xóa khi tắt nguồn.
- Mục đích: Đảm bảo tính bền vững (Persistence) cho dữ liệu của chương trình, cho phép dữ liệu tồn tại và được truy cập sau khi chương trình hoặc hệ thống đã tắt.

## Text files and lines

- Tập là Dãy các Dòng: Tập văn bản là một dãy (sequence) các dòng, giống như chuỗi là dãy các ký tự.
- Ký Tự Xuống Dòng (\n): Mỗi dòng kết thúc bằng một ký tự đặc biệt gọi là ký tự xuống dòng (\n).
- Tính chất \n: Mặc dù trông như hai ký tự (backslash và n), \n thực chất là một ký tự đơn lẻ, được Python sử dụng để ngắt dòng khi in hoặc hiển thị.

## Searching through a file

Mô hình phổ biến: Lặp qua tập, bỏ qua các dòng không cần thiết, và chỉ xử lý các dòng "thú vị".

- Lọc Điều kiện: Sử dụng phương thức chuỗi như `.startswith()` (lọc tiền tố) hoặc `.find()` (lọc chuỗi con bất kỳ).
- Xử lý Khoảng cách: Dùng `.rstrip()` để loại bỏ ký tự xuống dòng (\n) ở cuối mỗi dòng, tránh bị gián cách đôi khi in ra.
- Cấu trúc Nâng cao: Sử dụng lệnh `continue` để ngay lập tức bỏ qua các dòng không thỏa mãn điều kiện và chuyển sang lần lặp tiếp theo.

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if line.find('@uct.ac.za') == -1: continue
    print(line)
```

## Opening Files

Để truy cập dữ liệu bền vững được lưu trữ trên bộ nhớ thứ cấp, chương trình phải mở tệp đó bằng hàm Python `open('tên_tệp')`. Quá trình này giao tiếp với Hệ Điều Hành để xác định vị trí tệp. Nếu thành công, hàm sẽ trả về một File Handle (bộ xử lý tệp). Handle này không phải là dữ liệu của tệp, mà là một công cụ để chương trình thực hiện các thao tác như đọc, ghi và đóng tệp. Nếu tệp không tồn tại, thao tác `open()` sẽ thất bại và gây ra lỗi `FileNotFoundError`.

## Reading files

### 1. Đọc Từng Dòng bằng Vòng Lặp for

- Cơ chế: Dùng vòng lặp `for` để duyệt qua file handle.
  - Cú pháp: `for line in fhand:`
- Hoạt động: Vòng lặp `for` xử lý việc tách dữ liệu thành các dòng riêng biệt, mỗi dòng bao gồm cả ký tự xuống dòng (\n) ở cuối.
- Hiệu quả: Phương pháp này hiệu quả vì nó chỉ đọc từng dòng một (one line at a time), giúp chương trình đếm hoặc xử lý dữ liệu trong các tệp rất lớn (thậm chí gigabyte) mà không làm đầy bộ nhớ chính (Main Memory).

### 2. Đọc Toàn Bộ Tệp bằng Phương Thức .read()

- Phương thức: Dùng phương thức `.read()` trên file handle.
  - Cú pháp: `inp = fhand.read()`
- Hoạt động: Phương thức này đọc toàn bộ nội dung của tệp vào một chuỗi lớn duy nhất. Tất cả ký tự, bao gồm cả các ký tự xuống dòng (\n), được lưu trữ trong biến chuỗi đó.
- Lưu ý: Chỉ nên sử dụng phương pháp này khi bạn biết chắc chắn rằng kích thước tệp tương đối nhỏ so với dung lượng bộ nhớ chính của máy tính.

## Letting the user choose the file name

### 1. Mục Đích

Thay vì phải chỉnh sửa code Python mỗi lần muốn xử lý một tệp khác, chương trình nên cho phép người dùng nhập tên tệp khi chạy. Điều này giúp chương trình trở nên linh hoạt và có thể tái sử dụng trên nhiều tệp khác nhau mà không cần thay đổi code.

### 2. Kỹ Thuật

- Sử dụng hàm `input()` để đọc tên tệp từ người dùng và lưu vào một biến (ví dụ: `fname`).
- Truyền biến đó vào hàm `open()`.

## Using try, except, and open

- Mục đích: Ngăn chương trình bị lỗi (FileNotFoundError) và thoát đột ngột khi người dùng nhập tên tệp không tồn tại.
- Cơ chế:
  - 1.try: Đặt lệnh nguy hiểm (fhand = open(fname)) vào đây.
  - 2.except: Nếu open() thất bại, khối này sẽ được thực thi để xử lý lỗi một cách duyên dáng (ví dụ: in thông báo lỗi và dùng exit() để chấm dứt chương trình).

```
fname = input('Enter the file name: ')
try:
    fhand = open(fname)
except:
    print('File cannot be opened:', fname)
    exit()
count = 0
for line in fhand:
    if line.startswith('Subject:'):
        count = count + 1
print('There were', count, 'subject lines in', fname)
```

## Debugging

- Vấn đề Khoảng trắng: Các ký tự vô hình (space, \t, \n) gây khó khăn khi gỡ lỗi.
- Giải pháp: Dùng hàm repr() để hiển thị rõ ràng các chuỗi thoát ngược (\n, \t).
- Vấn đề Xuống Dòng Đa Hệ Thống: Các hệ thống khác nhau dùng các ký tự xuống dòng khác nhau (\n, \r, hoặc cả hai), gây lỗi khi di chuyển tệp.
- Lưu ý: Cần chuyển đổi định dạng tệp nếu di chuyển giữa các hệ thống khác nhau.

## Writing files

- Mở: Phải mở tệp bằng chế độ 'w' (write).
- Lưu ý: Nếu tệp đã tồn tại, nội dung cũ sẽ bị xóa sạch. Nếu tệp không tồn tại, tệp mới sẽ được tạo.
- Ghi: Dùng phương thức .write(string) để đưa dữ liệu vào tệp.
- Xuống Dòng: Phương thức .write() không tự động thêm ký tự xuống dòng; bạn phải tự thêm \n vào chuỗi.
- Đóng: Bắt buộc phải gọi .close() sau khi ghi xong để đảm bảo dữ liệu cuối cùng được lưu vật lý vào đĩa.

## Glossary

Thuật Ngữ	Định Nghĩa
<b>catch</b> (Bắt lỗi)	Ngăn chặn một ngoại lệ (exception) làm chấm dứt chương trình bằng cách sử dụng các câu lệnh <b>try</b> và <b>except</b> .
<b>newline</b> (Ký tự xuống dòng)	Một ký tự đặc biệt (\n) được sử dụng trong tệp và chuỗi để <b>chỉ ra kết thúc một dòng</b> .

# Chương 8: LISTS

## List là một dãy (sequence)

- List = tập hợp các giá trị (elements/items).
- Tạo bằng ngoặc vuông []:

```
nums = [10, 20, 30]; mix = ['spam', 2.0, 5, [10, 20]]
```

## Tính chất mutable (có thể thay đổi)

- Truy cập phần tử: `cheeses[0]`
- Gán giá trị mới:

```
numbers = [17, 123]; numbers[1] = 5
```

## Duyệt list

```
for x in [1,2,3]: print(x)
```

Hoặc duyệt theo chỉ số:

```
for i in range(len(list)): ...
```

## Các phép toán trên list

- Cộng: `[1,2] + [3,4] → [1,2,3,4]`
- Nhân: `[1]*3 → [1,1,1]`
- in: kiểm tra phần tử có trong list.
- So sánh bằng (==) kiểm tra giá trị, is kiểm tra cùng đối tượng không.

## Cắt list (slice)

```
t[1:3] # Lấy phần tử từ 1 đến 2  
t[:2] # Đầu → 1  
t[2:] # 2 → cuối  
t[:] # sao chép toàn bộ
```

## Các hàm (methods) phổ biến

```
t.append(x) # Thêm phần tử  
t.extend(l) # Nối list khác  
t.sort() # Sắp xếp  
t.reverse() # Đảo ngược
```

## Xóa phần tử

```
del t[i]      # Xóa theo vị trí  
t.pop(i)     # Xóa và trả về phần tử  
t.remove('x') # Xóa phần tử theo giá trị  
del t[1:3]    # Xóa nhiều phần tử
```

## List và hàm

len(), max(), min(), sum(), sorted()

Ví dụ tính trung bình:

```
avg = sum(nums) / len(nums)
```

## List và chuỗi

list(s) → chuyển chuỗi thành list ký tự.  
''.join(list) → nối list thành chuỗi.  
split() → tách chuỗi thành list.

## Phân tích dòng (Parsing)

Dùng .split() để chia dòng văn bản thành từ:

```
line.split()
```

## Đối tượng & Giá trị

Hai list có cùng giá trị nhưng khác đối tượng.

Dùng is để kiểm tra có cùng đối tượng không.

## Aliasing

```
a = [1,2,3]; b = a
```

```
b[0] = 99 # thay đổi a luôn
```

→ Tránh alias nếu không muốn ảnh hưởng chéo.

## List trong hàm

Truyền list → truyền tham chiếu (tham biến).

append() thay đổi gốc, còn + tạo list mới.

Ví dụ:

```
def delete_head(t): del t[0]
```

## Debugging (lỗi hay gặp)

Nhiều hàm thay đổi list và trả về  
None

```
t = t.sort() # sai → t thành None
```

```
t.sort()    # đúng
```

# Chương 9: DICTIONARIES

## Dictionary là gì?

-Dictionary (dict): Tập hợp các cặp key-value.

-Tạo bằng {}:

```
eng2vn = {'one':'một', 'two':'hai'}  
print(eng2vn['one']) # → một
```

Key: duy nhất, không thay đổi (immutable).  
Value: có thể là bất kỳ kiểu dữ liệu nào.

## Dùng như bộ đếm (Set of counters)

Đếm số lần xuất hiện của từ:

```
words = ['a', 'b', 'a']  
counts = {}  
for w in words:  
    counts[w] = counts.get(w, 0) + 1  
get(key, default) giúp tránh lỗi  
KeyError.
```

## Lặp qua dictionary

```
for key, value in counts.items():  
    print(key, value)
```

-keys(), values(), items() trả về danh sách tương ứng.

-Duyệt có thể theo key, value, hoặc cặp key-value.

## Tạo và cập nhật dictionary

```
counts = dict()  
counts['apple'] = 3  
counts['banana'] = 1  
counts['apple'] += 1  
print(counts) # {'apple':4, 'banana':1}
```

Kiểm tra key:

```
'apple' in counts # True
```

Duyệt qua dict:

```
for key in counts:  
    print(key, counts[key])
```

## Dictionary và file

Đếm từ trong file:

```
fhand = open('words.txt')  
counts = {}  
for line in fhand:  
    for word in line.split():  
        counts[word] = counts.get(word, 0) + 1
```

## Debugging

- Cẩn thận lỗi KeyError khi truy cập key không tồn tại → dùng get().
- Dictionary không có thứ tự (unordered).
- Không dùng list làm key vì list có thể thay đổi.

## Phân tích văn bản nâng cao (Advanced parsing)

Kết hợp .split() và dictionary để đếm, tìm từ xuất hiện nhiều nhất:

```
bigword = None
bigcount = None
for word,count in counts.items():
if bigcount is None or count > bigcount:
    bigword = word
    bigcount = count
print(bigword, bigcount)
```

## Các hàm hữu ích

Hàm	Ý nghĩa
len(d)	số phần tử
d.keys()	danh sách key
d.values()	danh sách value
d.items()	danh sách cặp (key, value)
d.get(k,0)	trả về giá trị hoặc mặc định
del d[k]	xóa key