

ĐỀ CƯƠNG ÔN TẬP

Cơ sở lập trình

Dành cho sinh viên đại học khối kinh tế, tài chính — ngôn ngữ Python

Principles of Programming — a bilingual study guide in Python

Thuật toán và môi trường lập trình · Biến và kiểu dữ liệu · Rẽ nhánh · Vòng lặp · Hàm và
tầm vực · Độ quy · Mảng và danh sách · Kiểu dữ liệu có cấu trúc · Tập, lỗi và gỡ lỗi

Một ứng dụng quản lý tài chính cá nhân xây dần qua từng chương

Mọi đoạn mã đã chạy thật · Bài tập có lời giải · Tự luận có dàn ý · Trắc nghiệm có đáp án

Song ngữ Việt – Anh · Vietnamese – English

Phiên bản 1.0 — Tháng 10 năm 2026

<https://globalmarket.vn>

M U C L U C

CONTENTS

Chương 1. Thuật toán, chương trình và môi trường lập trình Python — Chapter 1.	
<i>Algorithms, programs and the Python environment</i>.....	5
1.1. Thuật toán — <i>Algorithms</i>	5
1.2. Biểu diễn thuật toán — <i>Representing algorithms</i>	8
1.3. Ngôn ngữ lập trình và cách thực thi chương trình — <i>Programming languages and how programs run</i>	12
1.4. Đặc điểm của Python — <i>Features of Python</i>	14
1.5. Cài đặt Python và Visual Studio Code — <i>Installing Python and Visual Studio Code</i>	15
1.6. Solution, project và thư mục làm việc trong VS Code — <i>Solution, project and the working folder in VS Code</i>	16
1.7. Chạy chương trình: tệp .py và REPL — <i>Running programs: .py files and the REPL</i> ..	17
1.8. Cấu trúc tệp chương trình, chú thích, thụt lề và PEP 8 — <i>File structure, comments, indentation and PEP 8</i>	19
1.9. Ba loại lỗi: cú pháp, lúc chạy, logic — <i>Three kinds of error: syntax, runtime, logic</i> ..	22
1.10. Sổ Xu, bản 0.1 — <i>Sổ Xu, version 0.1</i>	25
1.11. Bài tập lập trình có lời giải — <i>Programming exercises with solutions</i>	26
1.12. Lỗi hay gặp khi bắt đầu với Python — <i>Common mistakes when starting with Python</i>	30
1.13. Bảng thuật ngữ — <i>Glossary</i>	31
1.14. Câu hỏi tự luận và dàn ý trả lời — <i>Essay questions and answer outlines</i>	33
1.15. Câu hỏi trắc nghiệm — <i>Multiple-choice questions</i>	34
Chương 2. Biến, hằng, kiểu dữ liệu và biểu thức — Chapter 2. Variables, constants, data types and expressions	36
2.1. Biến và phép gán — <i>Variables and assignment</i>	36
2.2. Quy tắc đặt tên, từ khóa và hằng — <i>Naming rules, keywords and constants</i>	38
2.3. Kiểu dữ liệu cơ bản — <i>Basic data types</i>	42
2.4. Toán tử số học và thứ tự ưu tiên — <i>Arithmetic operators and precedence</i>	44
2.5. Số thực dấu phẩy động, round và Decimal — <i>Floating-point numbers, round and Decimal</i>	49
2.6. Ép kiểu, input và print — <i>Type conversion, input and print</i>	51
2.7. Chuỗi định dạng f-string — <i>Formatted string literals (f-strings)</i>	54
2.8. Chuỗi cơ bản — <i>Basic strings</i>	56
2.9. Sổ Xu bản 0.2: nhập một khoản chi và tính số dư — <i>Sổ Xu version 0.2: entering one expense and computing the balance</i>	61
2.10. Bài tập lập trình có lời giải — <i>Worked programming exercises</i>	63
2.11. Bảng thuật ngữ — <i>Glossary</i>	66
2.12. Câu hỏi tự luận và dàn ý trả lời — <i>Essay questions and answer outlines</i>	68
2.13. Câu hỏi trắc nghiệm — <i>Multiple-choice questions</i>	69

Chương 3. Cấu trúc rẽ nhánh — Chapter 3. Selection structures.....	72
3.1. Giá trị logic và phép so sánh — <i>Boolean values and comparison.....</i>	72
3.2. Toán tử logic và giá trị truthy, falsy — <i>Logical operators and truthy and falsy values</i>	74
3.3. Câu lệnh if và if-else — <i>The if and if-else statements</i>	77
3.4. if-elif-else và biểu thức điều kiện — <i>if-elif-else and the conditional expression.....</i>	80
3.5. Câu lệnh match-case — <i>The match-case statement.....</i>	83
3.6. Kiểm tra dữ liệu nhập — <i>Validating input.....</i>	85
3.7. Ứng dụng trong Sổ Xu — <i>Application in Sổ Xu</i>	86
3.8. Lỗi hay gặp khi viết rẽ nhánh — <i>Common mistakes in selection code.....</i>	91
3.9. Bảng thuật ngữ — <i>Glossary.....</i>	93
3.10. Bài tập lập trình có lời giải — <i>Programming exercises with solutions.....</i>	95
3.11. Câu hỏi tự luận và dàn ý trả lời — <i>Essay questions and answer outlines</i>	100
3.12. Câu hỏi trắc nghiệm — <i>Multiple-choice questions</i>	101
Chương 4. Cấu trúc lặp — Chapter 4. Repetition structures	103
4.1. Khái niệm vòng lặp — <i>The idea of repetition.....</i>	103
4.2. Vòng lặp while — <i>The while loop.....</i>	104
4.3. Vòng lặp for và hàm range — <i>The for loop and the range function.....</i>	107
4.4. break, continue và else của vòng lặp — <i>break, continue and the loop else clause...110</i>	
4.5. Vòng lặp lồng nhau — <i>Nested loops.....</i>	113
4.6. Các mẫu thuật toán với vòng lặp — <i>Algorithm patterns with loops</i>	116
4.7. Vòng lặp vô hạn và cách tránh — <i>Infinite loops and how to avoid them</i>	119
4.8. Sổ Xu bản 0.4: tiết kiệm, trả góp và menu — <i>Sổ Xu version 0.4: savings, instalments</i> <i>and menu</i>	121
4.9. Đếm số lần lặp và độ phức tạp $O(n)$, $O(n^2)$ — <i>Counting iterations and $O(n)$, $O(n^2)$</i> <i>complexity.....</i>	124
4.10. Bảng thuật ngữ — <i>Glossary.....</i>	126
4.11. Bài tập lập trình có lời giải — <i>Worked programming exercises</i>	128
4.12. Câu hỏi tự luận và dàn ý trả lời — <i>Essay questions and answer outlines</i>	133
4.13. Câu hỏi trắc nghiệm — <i>Multiple-choice questions</i>	134
Chương 5. Hàm và tầm vực — Chapter 5. Functions and scope.....	136
5.1. Định nghĩa hàm và lệnh def — <i>5.1. Defining a function with def.....</i>	136
5.2. Giá trị trả về — <i>5.2. Return values.....</i>	138
5.3. Tham số và đối số — <i>5.3. Parameters and arguments</i>	141
5.4. Số đối số không định trước và tham số chỉ từ khóa — <i>5.4. Variable numbers of</i> <i>arguments and keyword-only parameters</i>	143
5.5. Truyền đối tượng: kiểu bất biến và khả biến — <i>5.5. Passing objects: immutable and</i> <i>mutable types.....</i>	145
5.6. Tầm vực của biến và quy tắc LEGB — <i>5.6. Variable scope and the LEGB rule.....</i>	148
5.7. Docstring, chú thích kiểu và hàm lambda — <i>5.7. Docstrings, type hints and lambda</i> <i>functions</i>	152

5.8. Module, lệnh import và khối <code>__main__</code> — 5.8. Modules, import and the <code>__main__</code> block	154
5.9. Sổ Xu: tách chương trình thành các hàm — 5.9. Sổ Xu: splitting the program into functions	156
5.10. Lỗi thường gặp về hàm và tầm vực — 5.10. Common errors with functions and scope	161
5.11. Bảng thuật ngữ — 5.11. Glossary	162
5.12. Bài tập lập trình có lời giải — 5.12. Worked programming exercises	163
5.13. Câu hỏi tự luận và dàn ý trả lời — Essay questions and answer outlines	169
5.14. Câu hỏi trắc nghiệm — Multiple-choice questions	171
Chương 6. Đệ quy — Chapter 6. Recursion	173
6.1. Khái niệm đệ quy: trường hợp cơ sở và bước đệ quy — Recursion: the base case and the recursive step	173
6.2. Ngăn xếp lời gọi và hàm giai thừa — The call stack and the factorial function	175
6.3. Dãy Fibonacci: cây lời gọi và ghi nhớ — The Fibonacci sequence: the call tree and memoisation	178
6.4. Đệ quy trên số và chuỗi — Recursion on numbers and strings	181
6.5. Tháp Hà Nội — The Tower of Hanoi	183
6.6. Tìm kiếm nhị phân đệ quy — Recursive binary search	185
6.7. Sổ Xu: số dư tiết kiệm sau n tháng — Sổ Xu: the savings balance after n months	186
6.8. Đệ quy và vòng lặp; giới hạn đệ quy — Recursion versus iteration; the recursion limit	188
6.9. Bảng thuật ngữ — Glossary	192
6.10. Bài tập lập trình có lời giải — Programming exercises with solutions	193
6.11. Câu hỏi tự luận và dàn ý trả lời — Essay questions and answer outlines	197
6.12. Câu hỏi trắc nghiệm — Multiple-choice questions	199
Chương 7. Mảng và danh sách — Chapter 7. Arrays and lists	202
7.1. Mảng và danh sách — Arrays and lists	202
7.2. Khai báo, nhập, xuất và chỉ số — Creating, reading, printing and indexing	204
7.3. Cắt lát — Slicing	206
7.4. Thêm, chèn, xóa và kiểm tra thành viên — Adding, inserting, deleting and membership tests	207
7.5. Duyệt list — Traversing a list	209
7.6. Tìm kiếm tuần tự và tìm kiếm nhị phân — Linear search and binary search	210
7.7. Sắp xếp — Sorting	213
7.8. List comprehension — List comprehension	216
7.9. Mảng hai chiều (ma trận) — Two-dimensional arrays (matrices)	217
7.10. Tuple, sao chép nông và sao chép sâu — Tuples, shallow copy and deep copy	219
7.11. Sổ Xu: mảng số tiền chi — Sổ Xu: the array of spending amounts	221
7.12. Bài tập lập trình có lời giải — Worked programming exercises	223
7.13. Bảng thuật ngữ — Glossary	227
7.14. Câu hỏi tự luận và dàn ý trả lời — Essay questions and answer outlines	229

7.15. Câu hỏi trắc nghiệm — <i>Multiple-choice questions</i>	230
Chương 8. Kiểu dữ liệu có cấu trúc — Chapter 8. Structured data types	233
8.1. Bản ghi và kiểu dữ liệu có cấu trúc — 8.1. Records and structured data types	233
8.2. Bốn cách biểu diễn một bản ghi trong Python — 8.2. Four ways to represent a record in Python	234
8.3. Từ điển dict — 8.3. The dict type	239
8.4. Tập hợp set — 8.4. The set type	242
8.5. Danh sách bản ghi — 8.5. A list of records	246
8.6. Nhóm và tổng hợp: defaultdict và Counter — 8.6. Grouping and aggregating: defaultdict and Counter	249
8.7. Dữ liệu JSON — 8.7. JSON data	252
8.8. Sổ Xu: dataclass GiaoDich và báo cáo ngân sách — 8.8. Sổ Xu: the GiaoDich dataclass and the budget report	254
8.9. Bảng thuật ngữ — 8.9. Glossary	258
8.10. Bài tập lập trình có lời giải — 8.10. Programming exercises with solutions	259
8.11. Câu hỏi tự luận và dàn ý trả lời — Essay questions and answer outlines	263
8.12. Câu hỏi trắc nghiệm — Multiple-choice questions	265
Chương 9. Tập, xử lý lỗi, gỡ lỗi và dự án hoàn chỉnh — Chapter 9. Files, error handling, debugging and a complete project	267
9.1. Đọc và ghi tệp văn bản — Reading and writing text files	267
9.2. Tệp CSV và module csv — CSV files and the csv module	270
9.3. Xử lý ngoại lệ: try, except, else, finally — Exception handling: try, except, else, finally	273
9.4. Các ngoại lệ thường gặp — Common exceptions	277
9.5. Đọc traceback — Reading a traceback	280
9.6. Gỡ lỗi (debug) — Debugging	282
9.7. assert và kiểm thử đơn giản — assert and simple testing	285
9.8. Dự án Sổ Xu hoàn chỉnh — The complete Sổ Xu project	287
9.9. Bài tập lập trình có lời giải — Worked programming exercises	294
9.10. Bảng thuật ngữ — Glossary	299
9.11. Câu hỏi tự luận và dàn ý trả lời — Essay questions and answer outlines	301
9.12. Câu hỏi trắc nghiệm — Multiple-choice questions	303
Phụ lục A. Đáp án câu hỏi trắc nghiệm — Appendix A. Answer key	305
Phụ lục B. Bảng tra cú pháp Python — Appendix B. Python syntax reference	310
Phụ lục C. Thuật ngữ Việt – Anh — Appendix C. Vietnamese–English glossary	312
Phụ lục D. Hàm dựng sẵn và phương thức thường dùng — Appendix D. Common built-in functions and methods	320
Tài liệu tham khảo — References	322

Chương 1. Thuật toán, chương trình và môi trường lập trình Python — Chapter 1. Algorithms, programs and the Python environment

Mục tiêu chương — Chapter objectives

- Phát biểu định nghĩa thuật toán, năm tính chất của thuật toán và biểu diễn một thuật toán bằng ngôn ngữ tự nhiên, lưu đồ, mã giả.
State the definition of an algorithm and its five properties, and express an algorithm in natural language, as a flowchart and in pseudocode.
- Phân biệt ngôn ngữ máy, hợp ngữ, ngôn ngữ bậc cao; biên dịch và thông dịch; nêu đặc điểm của Python.
Distinguish machine language, assembly language and high-level languages, compilation and interpretation, and name the main features of Python.
- Cài Python và Visual Studio Code, chọn trình thông dịch, mở thư mục làm việc, chạy tệp .py trong terminal tích hợp và trong REPL.
Install Python and Visual Studio Code, select an interpreter, open a working folder and run a .py file in the integrated terminal and in the REPL.
- Viết chương trình đúng cấu trúc tệp, chú thích, thụt lề và quy ước PEP 8; nhận diện ba loại lỗi: cú pháp, lúc chạy, logic.
Write a program with a correct file structure, comments, indentation and PEP 8 conventions, and recognise the three kinds of error: syntax, runtime and logic.
- Chạy bản 0.1 của Sổ Xu: in lời chào và số dư cuối tháng.
Run version 0.1 of Sổ Xu, which prints a greeting and the end-of-month balance.

1.1. Thuật toán — Algorithms

Thuật toán (algorithm) là dãy hữu hạn các bước, mỗi bước được mô tả đủ rõ để thực hiện máy móc, nhận dữ liệu vào và cho ra kết quả của một bài toán. Người hoặc máy đều thực hiện được thuật toán. **Chương trình** (program) là thuật toán được viết bằng một ngôn ngữ lập trình để máy tính chạy.

*An **algorithm** is a finite sequence of steps, each described precisely enough to be carried out mechanically, that takes input data and produces the result of a problem. People or machines can carry out an algorithm. A **program** is an algorithm written in a programming language so that a computer can run it.*

Công thức nấu ăn, quy trình rút tiền ở máy ATM và cách tính số dư cuối tháng là các thuật toán. Một thuật toán đúng phải có đủ năm tính chất trong bảng sau.

A recipe, the procedure for withdrawing cash at an ATM and the way to compute an end-of-month balance are all algorithms. A correct algorithm has the five properties in the table below.

Tính chất <i>Property</i>	Nội dung <i>Meaning</i>	Ví dụ: thuật toán Euclid tìm ước chung lớn nhất <i>Example: Euclid's algorithm for the greatest common divisor</i>
Hữu hạn <i>Finiteness</i>	Dừng sau một số hữu hạn bước với mọi đầu vào hợp lệ. <i>Stops after a finite number of steps for every valid input.</i>	Số dư của phép chia giảm dần và không âm nên đến lúc bằng 0. <i>The remainder shrinks and is never negative, so it reaches 0.</i>
Xác định <i>Definiteness</i>	Mỗi bước không mơ hồ; cùng đầu vào luôn cho cùng kết quả. <i>Every step is unambiguous; the same input always gives the same result.</i>	Bước "lấy số dư của a chia b" chỉ có một cách hiểu. <i>The step "take the remainder of a divided by b" has one reading.</i>
Đúng <i>Correctness</i>	Cho kết quả đúng với mọi đầu vào hợp lệ. <i>Gives the right result for every valid input.</i>	Số cuối cùng khác 0 chia hết cả hai số ban đầu và là lớn nhất. <i>The last non-zero number divides both original numbers and is the largest such number.</i>
Có đầu vào và đầu ra <i>Input and output</i>	Có không hoặc nhiều đầu vào, có ít nhất một đầu ra. <i>Zero or more inputs and at least one output.</i>	Vào: hai số nguyên dương. Ra: một số nguyên dương. <i>In: two positive integers. Out: one positive integer.</i>
Hiệu quả <i>Effectiveness</i>	Mỗi thao tác đủ đơn giản để thực hiện được; thời gian và bộ nhớ ở mức chấp nhận được. <i>Each operation is simple enough to perform; time and memory stay acceptable.</i>	Chỉ dùng phép chia lấy dư và phép gán, ít bước hơn so với thử từng ước số. <i>Uses only the remainder operation and assignment, in fewer steps than trying every divisor.</i>

Đoạn mã dưới đây cài đặt thuật toán Euclid cho hai số 48 và 18. Vòng lặp "while" lặp lại khi b khác 0 (chapter 4 giải thích kỹ); phép gán "a, b = b, a % b" gán đồng thời hai biến. Mỗi lượt in ra giá trị mới để thấy phần dư giảm.

The code below implements Euclid's algorithm for 48 and 18. The "while" loop repeats while b is not 0 (chapter 4 explains it in detail); the assignment "a, b = b, a % b" assigns both variables at once. Each pass prints the new values so that the shrinking remainder is visible.

```
a, b = 48, 18
print("Đầu vào:", a, b)
while b != 0:
    a, b = b, a % b
    print("a =", a, "b =", b)
print("UCLN =", a)
```

Kết quả chạy / Output

```
Đầu vào: 48 18
a = 18 b = 12
a = 12 b = 6
a = 6 b = 0
UCLN = 6
```

Vòng lặp chạy 3 lượt rồi dừng khi b bằng 0; kết quả là 6. Phần dư 12, 6, 0 giảm dần, đó là lý do thuật toán hữu hạn.

The loop runs 3 times and stops when b becomes 0; the result is 6. The remainders 12, 6, 0 shrink, which is why the algorithm is finite.

Cách kiểm một thuật toán trên giấy là lập **bảng theo dõi biến** (trace table): mỗi hàng ghi giá trị các biến sau một bước. Bảng dưới theo dõi thuật toán Euclid và phải trùng với kết quả máy in ở trên.

*A way to check an algorithm on paper is a **trace table**: each row records the values of the variables after one step. The table below traces Euclid's algorithm and must agree with what the machine printed above.*

Bước Step	a	b	Ghi chú Note
0	48	18	Giá trị ban đầu <i>Initial values</i>
1	18	12	a mới = b cũ; b mới = $48 \% 18 = 12$ <i>new a = old b; new b = $48 \% 18 = 12$</i>
2	12	6	b mới = $18 \% 12 = 6$ <i>new b = $18 \% 12 = 6$</i>
3	6	0	b mới = $12 \% 6 = 0$, vòng lặp dừng <i>new b = $12 \% 6 = 0$, the loop stops</i>

Bài toán quen thuộc của Sổ Xu cũng là một thuật toán: lấy số dư đầu tháng, cộng tổng thu, trừ tổng chi. Ba số lấy từ dữ liệu tháng 9/2026 của Nguyễn Hoàng Minh.

A familiar Sổ Xu task is also an algorithm: take the opening balance, add total income and subtract total spending. The three numbers come from the September 2026 data of Nguyễn Hoàng Minh.

```
SO_DU_DAU_THANG = 2_500_000
tong_thu = 7_600_000
tong_chi = 3_134_000
so_du_cuoi_thang = SO_DU_DAU_THANG + tong_thu - tong_chi
print("Số dư cuối tháng:", so_du_cuoi_thang)
```

Kết quả chạy / Output

```
Số dư cuối tháng: 6966000
```

Python in số nguyên không có dấu phân cách hàng nghìn (6966000). Chương 2 dạy cách định dạng thành 6.966.000.

Python prints an integer without thousands separators (6966000). Chapter 2 shows how to format it as 6.966.000.

PHẢI THUỘC / MEMORISE — Thuật toán có năm tính chất: hữu hạn, xác định, đúng, có đầu vào và đầu ra, hiệu quả. Chương trình là thuật toán viết bằng ngôn ngữ lập trình.

An algorithm has five properties: finiteness, definiteness, correctness, input and output, effectiveness. A program is an algorithm written in a programming language.

1.2. Biểu diễn thuật toán — Representing algorithms

Có ba cách biểu diễn thuật toán thường dùng: ngôn ngữ tự nhiên, lưu đồ (flowchart) và mã giả (pseudocode). Cùng một thuật toán, mỗi cách phù hợp một mục đích khác nhau.

Three representations are in common use: natural language, flowcharts and pseudocode. For the same algorithm, each suits a different purpose.

Cách biểu diễn <i>Representation</i>	Ưu điểm <i>Strength</i>	Hạn chế <i>Weakness</i>
Ngôn ngữ tự nhiên <i>Natural language</i>	Dễ đọc với mọi người, không cần quy ước. <i>Easy for anyone to read, no conventions needed.</i>	Dễ mơ hồ, khó mô tả rẽ nhánh và lặp. <i>Easily ambiguous, hard for branching and repetition.</i>
Lưu đồ <i>Flowchart</i>	Thấy ngay thứ tự các bước và các nhánh. <i>The order of steps and the branches show at once.</i>	Tốn chỗ; bài dài vẽ rối. <i>Takes space; long programs become tangled.</i>
Mã giả <i>Pseudocode</i>	Gần với chương trình, chuyển sang Python gần như từng dòng. <i>Close to a program, converts to Python nearly line by line.</i>	Không có chuẩn chung, không chạy được. <i>No common standard, cannot be run.</i>

Ngôn ngữ tự nhiên — Natural language

Thuật toán tính số dư cuối tháng của Sổ Xu bằng lời, đánh số từng bước:

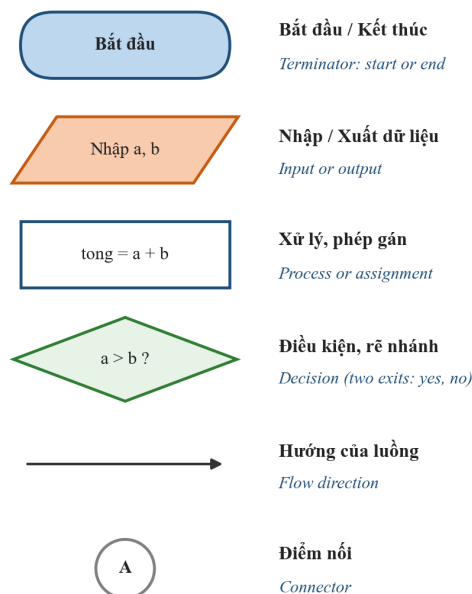
The algorithm for the Sổ Xu end-of-month balance, in words and numbered by step:

- Lấy số dư đầu tháng.
Take the opening balance.
- Cộng tổng thu của tháng vào số dư.
Add the month's total income to the balance.
- Trừ tổng chi của tháng khỏi kết quả.
Subtract the month's total spending from the result.
- Đưa kết quả ra màn hình rồi dừng.
Show the result on screen and stop.

Lưu đồ — Flowchart

Lưu đồ dùng các ký hiệu chuẩn nối với nhau bằng mũi tên. Hình 1.1 liệt kê sáu ký hiệu dùng trong sách này.

A flowchart joins standard symbols with arrows. Figure 1.1 lists the six symbols used in this book.



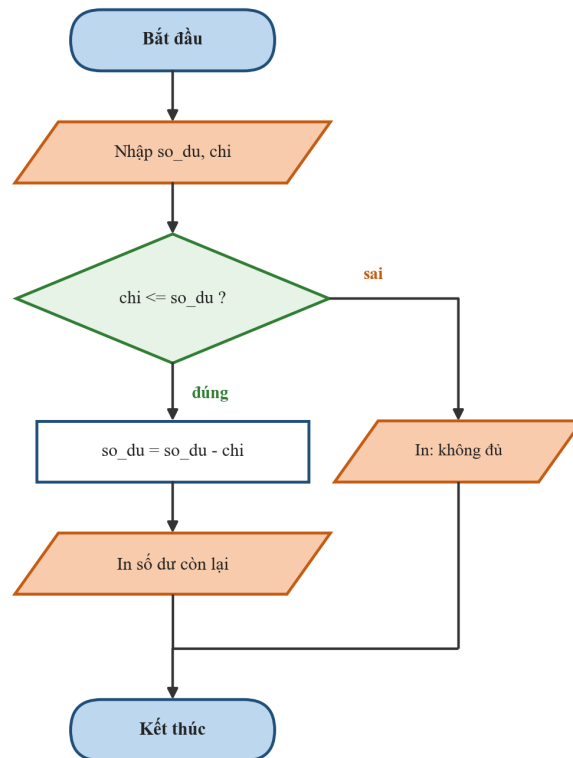
Hình 1.1. Các ký hiệu chuẩn của lưu đồ
Figure 1.1. Standard flowchart symbols

Quy tắc vẽ: mỗi lưu đồ có đúng một ký hiệu bắt đầu và một ký hiệu kết thúc; luồng đi từ trên xuống dưới; ký hiệu điều kiện có hai lối ra ghi "đúng" và "sai"; hai nhánh phải gặp lại nhau trước khi kết thúc.

Drawing rules: every flowchart has exactly one start and one end symbol; flow runs from top to bottom; a decision symbol has two exits labelled "yes" and "no"; the two branches must meet again before the end.

Hình 1.2 vẽ lưu đồ của một bài có rẽ nhánh: ghi một khoản chi vào Sổ Xu chỉ khi số dư đủ.

Figure 1.2 shows a flowchart with a branch: a spending entry is recorded in Sổ Xu only when the balance is enough.



Hình 1.2. Lưu đồ: ghi khoản chi nếu số dư đủ

Figure 1.2. Flowchart: record a spending entry if the balance is enough

Mã giả — Pseudocode

Mã giả viết các bước bằng câu lệnh ngắn gọn với lập trình nhưng không theo cú pháp của ngôn ngữ nào. Thụt lề thể hiện phần nằm trong NẾU. Mã giả của lưu đồ ở hình 1.2:

Pseudocode writes the steps as short statements close to programming but not following the syntax of any language. Indentation shows what lies inside IF. The pseudocode of the flowchart in figure 1.2:

```

BẮT ĐẦU
  NHẬP so_du, chi
  NẾU chi <= so_du THÌ
    so_du ← so_du - chi
    IN "Đã ghi khoản chi. Số dư còn lại:", so_du
  NGƯỢC LẠI
    IN "Không đủ số dư."
KẾT THÚC
  
```

Chuyển mã giả sang Python: NHẬP thành "input()" kèm "int()" để đổi chuỗi thành số nguyên, NẾU thành "if", NGƯỢC LẠI thành "else", dấu ← thành "=". Chạy với số dư 6.966.000 đồng và khoản chi 500.000 đồng:

Converting pseudocode to Python: NHẬP becomes "input()" with "int()" turning the text into an integer, NẾU becomes "if", NGƯỢC LẠI becomes "else", and the arrow ← becomes "=". Run with a balance of 6,966,000 dong and a spending entry of 500,000 dong:

```
so_du = int(input("Số dư hiện có: "))
chi = int(input("Khoản định chi: "))
if chi <= so_du:
    so_du = so_du - chi
    print("Đã ghi khoản chi. Số dư còn lại:", so_du)
else:
    print("Không đủ số dư.")
```

Kết quả chạy / Output

```
Số dư hiện có: 6966000
Khoản định chi: 500000
Đã ghi khoản chi. Số dư còn lại: 6466000
```

Chạy lại với số dư 300.000 đồng và khoản chi 450.000 đồng để đi qua nhánh "sai":

Run again with a balance of 300,000 dong and a spending entry of 450,000 dong to take the "no" branch:

```
so_du = int(input("Số dư hiện có: "))
chi = int(input("Khoản định chi: "))
if chi <= so_du:
    so_du = so_du - chi
    print("Đã ghi khoản chi. Số dư còn lại:", so_du)
else:
    print("Không đủ số dư.")
```

Kết quả chạy / Output

```
Số dư hiện có: 300000
Khoản định chi: 450000
Không đủ số dư.
```

Một bài rẽ nhánh khác: tìm khoản lớn nhất trong ba khoản chi theo danh mục của Sổ Xu (nhà ở 1.800.000, ăn uống 585.000, học tập 449.000). Mã giả:

Another branching task: find the largest of three spending categories of Sổ Xu (housing 1,800,000, food 585,000, study 449,000). Pseudocode:

```
BẮT ĐẦU
lon_nhat ← nha_o
NEẾU an_uong > lon_nhat THÌ lon_nhat ← an_uong
NEẾU hoc_tap > lon_nhat THÌ lon_nhat ← hoc_tap
IN lon_nhat
KẾT THÚC
```

```
nha_o = 1_800_000
an_uong = 585_000
hoc_tap = 449_000
lon_nhat = nha_o
if an_uong > lon_nhat:
    lon_nhat = an_uong
if hoc_tap > lon_nhat:
    lon_nhat = hoc_tap
print("Khoản chi lớn nhất:", lon_nhat)
```

Kết quả chạy / Output

```
Khoản chi lớn nhất: 1800000
```

PHẢI THUỘC / MEMORISE — Lưu đồ: hình bầu dục cho bắt đầu và kết thúc, hình bình hành cho nhập và xuất, hình chữ nhật cho xử lý, hình thoi cho điều kiện. Mã giả viết bằng câu lệnh ngắn, thật lè cho khối lệnh, không có cú pháp chuẩn.

Flowchart: an oval for start and end, a parallelogram for input and output, a rectangle for

processing, a diamond for a decision. Pseudocode uses short statements, indents blocks and has no standard syntax.

1.3. Ngôn ngữ lập trình và cách thực thi chương trình — *Programming languages and how programs run*

Bộ xử lý chỉ thực hiện được lệnh dạng nhị phân. Ngôn ngữ lập trình chia theo mức độ gần với máy hay gần với người thành ba nhóm.

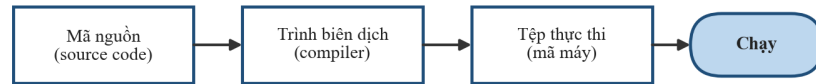
A processor can only carry out instructions in binary form. Programming languages fall into three groups, by how close they are to the machine or to the person.

Nhóm <i>Group</i>	Đặc điểm <i>Characteristics</i>	Ví dụ <i>Example</i>	Ưu điểm và hạn chế <i>Strength and weakness</i>
Ngôn ngữ máy <i>Machine language</i>	Dãy bit 0 và 1 mà bộ xử lý thực hiện trực tiếp. <i>Strings of 0 and 1 bits that the processor executes directly.</i>	Mã lệnh nhị phân của từng dòng bộ xử lý. <i>Binary opcodes of each processor family.</i>	Chạy nhanh nhất nhưng gần như không đọc nổi và gắn với một loại máy. <i>Fastest to run but almost unreadable and tied to one machine type.</i>
Hợp ngữ <i>Assembly language</i>	Dùng tên gọi nhớ thay cho mã nhị phân; mỗi lệnh ứng với một lệnh máy. <i>Uses mnemonic names in place of binary codes; each statement matches one machine instruction.</i>	Lệnh cộng thanh ghi có dạng ADD AX, BX. <i>A register addition looks like ADD AX, BX.</i>	Điều khiển được phần cứng nhưng dài, khó bảo trì. <i>Controls the hardware but is long and hard to maintain.</i>
Ngôn ngữ bậc cao <i>High-level language</i>	Gần ngôn ngữ con người, không phụ thuộc loại máy; một câu lệnh ứng với nhiều lệnh máy. <i>Close to human language, independent of machine type; one statement matches many machine instructions.</i>	Python, C, Java.	Viết nhanh, dễ đọc, chạy trên nhiều máy; cần chương trình dịch. <i>Quick to write, easy to read, runs on many machines; needs a translator.</i>

Máy chỉ hiểu mã máy nên mã nguồn bậc cao phải được dịch. Có hai cách dịch: **biên dịch** (compile) dịch toàn bộ chương trình thành tệp thực thi rồi mới chạy; **thông dịch** (interpret) dịch và thực hiện chương trình mỗi lần chạy. Hình 1.3 so sánh hai đường đi.

*The machine understands only machine code, so high-level source code has to be translated. There are two ways: **compiling** translates the whole program into an executable file before it runs; **interpreting** translates and executes the program every time it runs. Figure 1.3 compares the two paths.*

Biên dịch (ví dụ ngôn ngữ C)



Thông dịch (Python)



Biên dịch xong mới chạy, một lần cho cả chương trình

Trình thông dịch dịch và thực hiện mỗi lần chạy tệp .py

Hình 1.3. Biên dịch và thông dịch
Figure 1.3. Compiling and interpreting

Tiêu chí <i>Criterion</i>	Biên dịch <i>Compiling</i>	Thông dịch <i>Interpreting</i>
Thời điểm dịch <i>When translation happens</i>	Một lần, trước khi chạy. <i>Once, before running.</i>	Mỗi lần chạy chương trình. <i>Each time the program runs.</i>
Sản phẩm <i>Product</i>	Tập thực thi mã máy. <i>An executable machine-code file.</i>	Không tạo tệp thực thi riêng; kết quả hiện ngay. <i>No separate executable; the result appears at once.</i>
Phát hiện lỗi <i>Finding errors</i>	Lỗi cú pháp báo trước khi chạy. <i>Syntax errors reported before running.</i>	Lỗi cú pháp báo khi trình dịch đọc tới dòng lỗi; lỗi lúc chạy báo ở dòng gây ra. <i>Syntax errors are reported as the interpreter reads the faulty line; runtime errors at the line that causes them.</i>
Tốc độ <i>Speed</i>	Thường nhanh hơn. <i>Usually faster.</i>	Thường chậm hơn. <i>Usually slower.</i>
Sửa và chạy lại <i>Edit and rerun</i>	Phải dịch lại sau mỗi lần sửa. <i>Must recompile after each edit.</i>	Sửa xong chạy ngay. <i>Run straight after editing.</i>

Bản Python thông dụng nhất (CPython) làm hai việc: biên dịch mã nguồn sang bytecode (mã trung gian, lưu trong thư mục "__pycache__"), rồi máy ảo Python chạy bytecode đó. Người dùng không phải biên dịch riêng nên Python được xếp vào nhóm ngôn ngữ thông dịch.

The most common Python implementation (CPython) does two things: it compiles source code to bytecode (an intermediate code stored in the "__pycache__" folder), then the Python virtual

machine runs that bytecode. The user never compiles separately, so Python is classed as an interpreted language.

1.4. Đặc điểm của Python — Features of Python

- **Cú pháp ngắn:** một việc thường cần ít dòng hơn so với nhiều ngôn ngữ khác, không cần dấu chấm phẩy cuối dòng.
Short syntax: a task often needs fewer lines than in many other languages, and no semicolon is needed at the end of a line.
- **Thụt lề là cú pháp:** các dòng cùng khối lệnh phải thụt lề bằng nhau; không có cặp ngoặc nhọn bao khối.
Indentation is syntax: the lines of one block must be indented equally; there are no curly braces around blocks.
- **Kiểu động:** biến không khai báo kiểu, kiểu thuộc về giá trị đang gán cho biến.
Dynamic typing: variables are not declared with a type; the type belongs to the value currently assigned.
- **Thư viện chuẩn rộng:** có sẵn module cho toán, ngày giờ, tệp, CSV, JSON mà không cần cài thêm.
A wide standard library: modules for maths, dates, files, CSV and JSON come ready, with nothing to install.
- **Chạy trên nhiều hệ điều hành:** cùng một tệp .py chạy trên Windows, macOS, Linux.
Cross-platform: the same .py file runs on Windows, macOS and Linux.
- **Mã nguồn mở, miễn phí:** tải từ trang python.org.
Open source and free: downloadable from python.org.

Đoạn mã sau minh họa kiểu động: cùng một biến nhận giá trị số rồi giá trị chuỗi, không cần khai báo lại.

The following code shows dynamic typing: the same variable receives a number and then a string, with no new declaration.

```
gia_tri = 2_500_000
print(gia_tri)
gia_tri = "hai triệu rưỡi"
print(gia_tri)
```

Kết quả chạy / Output

```
2500000
hai triệu rưỡi
```

Python được dùng nhiều trong phân tích dữ liệu, tự động hóa và công nghệ tài chính, nên nhóm ngành này chọn làm ngôn ngữ nhập môn. Mọi ví dụ trong sách dùng Python 3.10 trở lên.

Python is widely used in data analysis, automation and financial technology, which is why this field takes it as the introductory language. Every example in this book uses Python 3.10 or later.

1.5. Cài đặt Python và Visual Studio Code — *Installing Python and Visual Studio Code*

Các bước cài đặt — *Installation steps*

1. Tải bản cài Python 3 từ python.org. Trên Windows, đánh dấu ô "Add python.exe to PATH" ở màn hình đầu của bộ cài để gõ được lệnh Python từ terminal.
Download the Python 3 installer from python.org. On Windows, tick "Add python.exe to PATH" on the first screen of the installer so that the Python command works from a terminal.
2. Mở terminal và kiểm tra bản cài (bảng bên dưới); lệnh in ra số phiên bản là đạt.
Open a terminal and check the installation (table below); a command that prints a version number means success.
3. Tải Visual Studio Code (VS Code) từ code.visualstudio.com và cài đặt.
Download Visual Studio Code (VS Code) from code.visualstudio.com and install it.
4. Mở VS Code, vào ngăn Extensions (biểu tượng bốn ô vuông ở thanh bên trái), tìm "Python" của Microsoft rồi chọn Install.
Open VS Code, go to the Extensions view (the four-squares icon on the left bar), search for the Microsoft "Python" extension and choose Install.
5. Chọn trình thông dịch: mở Command Palette (Ctrl+Shift+P, trên macOS là Cmd+Shift+P), gõ "Python: Select Interpreter" rồi chọn bản Python 3 đã cài. Tên bản chọn hiện ở góc phải thanh trạng thái.
Select the interpreter: open the Command Palette (Ctrl+Shift+P, Cmd+Shift+P on macOS), type "Python: Select Interpreter" and pick the Python 3 you installed. The chosen version appears at the right of the status bar.

Hệ điều hành <i>Operating system</i>	Kiểm tra phiên bản <i>Check the version</i>	Chạy tệp chao.py <i>Run chao.py</i>
Windows	py --version	py chao.py
macOS, Linux	python3 --version	python3 chao.py

Chương trình có thể kiểm tra phiên bản ngay từ mã. Điều kiện bên dưới so phiên bản đang chạy với 3.10, mốc cần cho câu lệnh "match-case" ở chương 3.

A program can check the version itself. The condition below compares the running version with 3.10, the minimum needed for the "match-case" statement in chapter 3.

```
import sys
print(sys.version_info >= (3, 10))
```

Kết quả chạy / *Output*

```
True
```

Kết quả True nghĩa là bản Python đang chạy từ 3.10 trở lên. Kết quả False nghĩa là cần cài bản mới hơn.

The result True means the running Python is 3.10 or later. The result False means a newer version must be installed.

1.6. Solution, project và thư mục làm việc trong VS Code — *Solution, project and the working folder in VS Code*

Chương trình đào tạo dùng thuật ngữ của Visual Studio: một **solution** gồm nhiều **project**, mỗi project gồm các tệp mã nguồn. VS Code không có hai khái niệm này; nó làm việc với thư mục. Bảng sau cho biết tương ứng.

*The curriculum uses Visual Studio terms: a **solution** holds several **projects**, and each project holds source files. VS Code has neither concept; it works with folders. The table gives the correspondence.*

Khái niệm quen dùng <i>Familiar concept</i>	Tương ứng trong Python và VS Code <i>Counterpart in Python and VS Code</i>	Ghi chú <i>Note</i>
Solution	Workspace: thư mục mở bằng File > Open Folder, hoặc tệp .code-workspace gom nhiều thư mục. <i>Workspace: a folder opened with File > Open Folder, or a .code-workspace file that groups several folders.</i>	Ví dụ: thư mục "CoSoLapTrinh" chứa mọi bài của học phần. <i>Example: the folder "CoSoLapTrinh" holds all work of the course.</i>
Project	Thư mục dự án: một thư mục con chứa các tệp .py của một ứng dụng. <i>Project folder: a subfolder holding the .py files of one application.</i>	Ví dụ: thư mục con "so_xu" chứa tệp "so_xu.py". <i>Example: the subfolder "so_xu" holds the file "so_xu.py".</i>
Tệp mã nguồn <i>Source file</i>	Tệp có đuôi .py. <i>A file ending in .py.</i>	Tên tệp viết thường, nối bằng dấu gạch dưới. <i>File names in lower case joined with underscores.</i>
Dựng (build) rồi chạy <i>Build then run</i>	Không có bước dựng riêng; chạy thẳng tệp .py. <i>No separate build step; run the .py file directly.</i>	Nút Run Python File hoặc lệnh trong terminal. <i>The Run Python File button or a terminal command.</i>
Gỡ lỗi (debug) <i>Debug</i>	Run and Debug trong VS Code, phím F5. <i>Run and Debug in VS Code, key F5.</i>	Chương 9 hướng dẫn điểm dừng. <i>Chapter 9 covers breakpoints.</i>
struct	dict, namedtuple hoặc dataclass. <i>dict, namedtuple or dataclass.</i>	Chương 8. <i>Chapter 8.</i>

Cây thư mục sau là cách tổ chức đề nghị cho cả học phần; cột trái là thư mục làm việc, các thư mục con là project:

The folder tree below is the suggested layout for the whole course; the top folder is the working folder and the subfolders are projects:

CoSoLapTrinh/	(workspace: mở bằng Open Folder)
chuong01/	(project: bài tập chương 1)
chao.py	
bai_tap_1.py	
so_xu/	(project: ứng dụng Sổ Xu)
so_xu.py	

Mở thư mục làm việc và terminal tích hợp — *Opening the working folder and the integrated terminal*

1. Tạo thư mục "CoSoLapTrinh" bằng trình quản lý tệp của hệ điều hành.
Create the folder "CoSoLapTrinh" with the operating system's file manager.
2. Trong VS Code chọn File > Open Folder rồi chọn thư mục vừa tạo; chọn Yes khi VS Code hỏi có tin cậy tác giả của thư mục.
In VS Code choose File > Open Folder and select that folder; answer Yes when VS Code asks whether you trust the authors of the folder.
3. Trong ngăn Explorer, dùng biểu tượng New File để tạo tệp "chao.py".
In the Explorer view, use the New File icon to create the file "chao.py".
4. Mở terminal tích hợp bằng Terminal > New Terminal. Terminal mở sẵn đúng thư mục làm việc nên gõ được lệnh chạy tệp mà không cần đổi thư mục.
Open the integrated terminal with Terminal > New Terminal. It opens in the working folder, so a command to run a file can be typed without changing folder.

1.7. Chạy chương trình: tệp .py và REPL — *Running programs: .py files and the REPL*

Có hai cách chạy Python. Cách thứ nhất là viết lệnh vào tệp .py rồi chạy cả tệp. Chương trình đầu tiên, lưu thành "chao.py":

There are two ways to run Python. The first is to write statements in a .py file and run the whole file. The first program, saved as "chao.py":

```
print("Xin chào, Sổ Xu!")
print("Số dư đầu tháng:", 2_500_000)
```

Kết quả chạy / Output

```
Xin chào, Sổ Xu!
Số dư đầu tháng: 2500000
```

Chạy bằng một trong ba cách: gõ lệnh trong terminal tích hợp (bảng ở mục 1.5); bấm nút hình tam giác Run Python File ở góc phải trên của cửa sổ soạn thảo; hoặc nhấn Ctrl+F5 để chạy không gỡ lỗi. Lệnh trong terminal, chạy ở thư mục chứa tệp:

Run it in one of three ways: type the command in the integrated terminal (table in section 1.5); click the triangle Run Python File button at the top right of the editor; or press Ctrl+F5 to run without debugging. The terminal command, run in the folder that holds the file:

```
python3 chao.py
```

Cách thứ hai là **REPL** (Read–Evaluate–Print Loop): gõ python3 (Windows: py) không kèm tên tệp để vào chế độ tương tác, dấu nhắc ">>>" xuất hiện; mỗi lệnh gõ xong được thực hiện ngay và giá trị của biểu thức được in lại. Thoát bằng "exit()". Bản ghi một phiên REPL, các giá trị bên dưới do Python tính khi dừng sách:

The second way is the **REPL** (Read–Evaluate–Print Loop): type `python3` (Windows: `py`) without a file name to enter interactive mode, where the `>>>` prompt appears; each statement runs as soon as it is typed and the value of an expression is printed back. Leave with `exit()`. A recorded REPL session, with the values below computed by Python when the book was built:

```
>>> so_du = 2_500_000
>>> so_du + 7_600_000 - 3_134_000
6966000
>>> 7 // 2
3
>>> 7 % 2
1
>>> len('Sổ Xu')
5
>>> 'Sổ' + ' Xu'
'Sổ Xu'
```

Tiêu chí <i>Criterion</i>	Tệp .py <i>.py file</i>	REPL
Dùng khi <i>Use for</i>	Chương trình hoàn chỉnh, chạy lại nhiều lần. <i>Complete programs run many times.</i>	Thử nhanh một biểu thức hay một lệnh. <i>Trying one expression or statement quickly.</i>
Lưu mã <i>Keeping code</i>	Mã được lưu trong tệp. <i>The code is saved in the file.</i>	Đóng REPL là mất lệnh đã gõ. <i>Closing the REPL loses the statements typed.</i>
In giá trị <i>Printing values</i>	Phải gọi <code>print()</code> mới thấy giá trị. <i>print() must be called to see a value.</i>	Biểu thức tự được in lại. <i>An expression is echoed automatically.</i>

Lệnh terminal thường dùng — *Common terminal commands*

Terminal chỉ chạy được tệp khi đang đứng đúng thư mục. Bốn lệnh sau đủ để di chuyển và kiểm tra vị trí.

The terminal can run a file only from the right folder. The four commands below are enough to move around and check the location.

Việc cần làm <i>Task</i>	macOS, Linux	Windows (Command Prompt)
Xem thư mục hiện tại <i>Show the current folder</i>	<code>pwd</code>	<code>cd</code>
Liệt kê tệp trong thư mục <i>List the files in the folder</i>	<code>ls</code>	<code>dir</code>
Vào thư mục con <i>Enter a subfolder</i>	<code>cd chuong01</code>	<code>cd chuong01</code>
Lên một cấp <i>Go up one level</i>	<code>cd ..</code>	<code>cd ..</code>

Phím tắt hay dùng trong VS Code — Useful VS Code shortcuts

Việc cần làm <i>Task</i>	Windows, Linux	macOS
Mở Command Palette <i>Open the Command Palette</i>	Ctrl+Shift+P	Cmd+Shift+P
Lưu tệp <i>Save the file</i>	Ctrl+S	Cmd+S
Chạy không gỡ lỗi <i>Run without debugging</i>	Ctrl+F5	Control+F5
Bật hoặc tắt chú thích cho dòng đang chọn <i>Toggle a comment on the selected lines</i>	Ctrl+/ 	Cmd+/
Thụt lề hoặc lùi lề dòng đang chọn <i>Indent or outdent the selected lines</i>	Tab, Shift+Tab	Tab, Shift+Tab

Vòng làm việc thông thường gồm bốn việc lặp lại: viết vài dòng, lưu tệp, chạy, đọc kết quả hoặc thông báo lỗi rồi sửa. Chạy sau mỗi vài dòng thay vì viết cả chương trình mới chạy, vì khi có lỗi thì chỉ phải tìm trong vài dòng vừa viết.

The usual working cycle has four repeated actions: write a few lines, save the file, run it, read the result or the error message and fix. Run after every few lines instead of writing the whole program first, because when an error appears it has to be searched for only in the few lines just written.

VÍ DỤ / EXAMPLE — Quy trình một bài thực hành: tạo tệp `bai_tap_1.py` trong thư mục `chuong01`; viết phần nhập; chạy thử với một bộ số; viết phần tính; chạy lại; viết phần in; chạy với số tính tay để đối chiếu.

The routine for one exercise: create `bai_tap_1.py` in the folder `chuong01`; write the input part; run it with one set of numbers; write the calculation; run again; write the output; run with hand-calculated numbers to compare.

1.8. Cấu trúc tệp chương trình, chú thích, thụt lề và PEP 8 — File structure, comments, indentation and PEP 8

Python thực hiện lệnh theo thứ tự từ trên xuống dưới, mỗi lệnh một dòng. Một tệp chương trình gọn thường theo thứ tự: dòng mô tả đầu tệp, các lệnh "import", các hằng, định nghĩa hàm (từ chương 5), rồi phần chạy chính.

Python executes statements from top to bottom, one per line. A tidy program file usually follows this order: a description at the top, the "import" statements, constants, function definitions (from chapter 5), then the main part.

Thứ tự chạy là thứ tự dòng trong tệp. Đổi chỗ hai dòng thì kết quả in ra đổi theo, và nếu dòng dùng biến đứng trước dòng gán biến thì Python báo `NameError`.

The running order is the order of lines in the file. Swapping two lines swaps the printed output, and if a line that uses a variable comes before the line that assigns it, Python raises `NameError`.

```
print("Bước 1: đọc dữ liệu")
print("Bước 2: tính toán")
print("Bước 3: in kết quả")
```

Kết quả chạy / Output

```
Bước 1: đọc dữ liệu
Bước 2: tính toán
Bước 3: in kết quả
```

```
print("Số dư:", so_du)
so_du = 6_966_000
```

Kết quả chạy / Output

```
Traceback (most recent call last):
  File "chuong_trinh.py", line 1, in <module>
    print("Số dư:", so_du)
                      ^^^^^
NameError: name 'so_du' is not defined
```

Chú thích — Comments

Chú thích (comment) bắt đầu bằng dấu "#" và kéo dài đến hết dòng; Python bỏ qua. Dòng mô tả đầu tệp đặt trong ba dấu nháy kép (docstring của module). Chú thích giải thích lý do, không nhắc lại điều mã đã nói rõ.

*A **comment** starts with "#" and runs to the end of the line; Python ignores it. The description at the top of a file goes between triple double quotes (the module docstring). A comment explains the reason and does not repeat what the code already says.*

```
"""Số Xu 0.1 - tính số dư cuối tháng."""
# Dữ liệu tháng 9/2026
SO_DU_DAU_THANG = 2_500_000 # đồng, ngày 01/9/2026
tong_thu = 7_600_000
# print("dòng này bị Python bỏ qua")
tong_chi = 3_134_000
print(SO_DU_DAU_THANG + tong_thu - tong_chi)
```

Kết quả chạy / Output

```
6966000
```

Thụt lề — Indentation

Sau dấu hai chấm ở cuối lệnh "if", "for", "while", "def", các dòng thuộc khối lệnh thụt vào cùng một mức, quy ước bốn dấu cách. Dòng trở lại lề cũ nằm ngoài khối. Trong đoạn sau, lệnh "print" cuối cùng nằm ngoài khối "if" nên luôn chạy:

After the colon at the end of an "if", "for", "while" or "def" line, the lines of the block are indented to the same level, by convention four spaces. A line back at the old margin is outside the block. In the following code the last "print" is outside the "if" block and therefore always runs:

```
so_du = 800_000
khoan_chi = 350_000
if khoan_chi <= so_du:
    so_du = so_du - khoan_chi
    print("Đã chi", khoan_chi)
print("Số dư:", so_du)
```

Kết quả chạy / Output

```
Đã chi 350000
Số dư: 450000
```

Thụt lề sai là lỗi cú pháp. Khối sau dấu hai chấm thiếu dòng thụt vào:

Wrong indentation is a syntax error. The block after the colon has no indented line:

```
so_du = 800_000
if so_du > 0:
print("Còn tiền")
```

Kết quả chạy / Output

```
File "chuong_trinh.py", line 3
    print("Còn tiền")
    ^^^^^
IndentationError: expected an indented block after 'if' statement on line 2
```

Dòng thụt vào khi không có lệnh mở khối cũng bị từ chối:

An indented line with no statement opening a block is also rejected:

```
so_du = 800_000
    print(so_du)
```

Kết quả chạy / Output

```
File "chuong_trinh.py", line 2
    print(so_du)
IndentationError: unexpected indent
```

PEP 8 — PEP 8

PEP 8 là hướng dẫn phong cách viết mã Python do cộng đồng Python soạn. Mã sai PEP 8 vẫn chạy, nhưng khó đọc và khó bảo trì. Bảng sau tóm tắt các quy tắc dùng trong sách.

PEP 8 is the style guide for Python code written by the Python community. Code that breaks PEP 8 still runs, but it is harder to read and maintain. The table summarises the rules used in this book.

Nội dung <i>Topic</i>	Quy tắc <i>Rule</i>	Ví dụ đúng <i>Good example</i>
Tên biến, hàm <i>Variable and function names</i>	Chữ thường, các từ nối bằng dấu gạch dưới (snake_case). <i>Lower case, words joined by underscores (snake_case).</i>	so_du_cuoi_thang
Hằng <i>Constants</i>	Chữ hoa, nối bằng dấu gạch dưới. <i>Upper case joined by underscores.</i>	SO_DU_DAU_THANG
Thụt lề <i>Indentation</i>	Bốn dấu cách mỗi mức, không trộn tab. <i>Four spaces per level, no mixed tabs.</i>	if a > b:
Độ dài dòng <i>Line length</i>	Tối đa 79 ký tự; dòng dài ngắt trong cặp ngoặc. <i>At most 79 characters; break long lines inside brackets.</i>	tong = (a + b + c)

Nội dung <i>Topic</i>	Quy tắc <i>Rule</i>	Ví dụ đúng <i>Good example</i>
Khoảng trắng <i>Spaces</i>	Một dấu cách hai bên toán tử và sau dấu phẩy. <i>One space around operators and after commas.</i>	tong = a + b
Một lệnh một dòng <i>One statement per line</i>	Không nối nhiều lệnh bằng dấu chấm phẩy. <i>Do not join statements with semicolons.</i>	a = 1

Hai đoạn dưới cùng làm một việc. Đoạn đầu chạy được nhưng vi phạm PEP 8 ở tên, khoảng trắng và dấu chấm phẩy; đoạn sau sửa theo PEP 8.

The two snippets below do the same thing. The first runs but breaks PEP 8 in names, spacing and semicolons; the second follows PEP 8.

```
SoDuDau=2500000;TongThu=7600000
print (SoDuDau+TongThu)
```

Kết quả chạy / Output

```
10100000
```

```
so_du_dau = 2_500_000
tong_thu = 7_600_000
print(so_du_dau + tong_thu)
```

Kết quả chạy / Output

```
10100000
```

Biểu thức dài được ngắt dòng bên trong cặp ngoặc, dòng tiếp theo thụt vào thẳng hàng với phần mở đầu:

A long expression is broken inside brackets, with the next line aligned with the start of the expression:

```
chi_nha_an_di_lai = (1_800_000 + 45_000 + 230_000 + 310_000
                    + 120_000)
print(chi_nha_an_di_lai)
```

Kết quả chạy / Output

```
2505000
```

PHẢI THUỘC / MEMORISE — Chú thích bắt đầu bằng "#". Khối lệnh thụt lề bốn dấu cách sau dấu hai chấm. PEP 8: tên biến snake_case, hằng CHỮ_HOÀ, dòng không quá 79 ký tự.

A comment starts with "#". A block is indented four spaces after the colon. PEP 8: snake_case variable names, UPPER_CASE constants, lines of at most 79 characters.

1.9. Ba loại lỗi: cú pháp, lúc chạy, logic — Three kinds of error: syntax, runtime, logic

Lỗi trong chương trình chia thành ba loại, khác nhau ở thời điểm phát hiện và ở dấu hiệu.

Errors in a program fall into three kinds, which differ in when they are found and how they show.

Loại lỗi <i>Kind of error</i>	Phát hiện khi nào <i>When it is found</i>	Dấu hiệu <i>Sign</i>	Cách tìm và sửa <i>How to find and fix</i>
Lỗi cú pháp <i>Syntax error</i>	Trước khi chạy dòng nào, khi Python đọc tệp. <i>Before any line runs, as Python reads the file.</i>	Chương trình không chạy; báo SyntaxError hoặc IndentationError kèm số dòng. <i>The program does not run; SyntaxError or IndentationError with a line number.</i>	Đọc dòng được báo và dòng ngay trước nó. <i>Read the reported line and the line just before it.</i>
Lỗi lúc chạy <i>Runtime error</i>	Trong lúc chạy, tại lệnh gây lỗi. <i>While running, at the faulty statement.</i>	Chương trình dừng giữa chừng và in traceback. <i>The program stops midway and prints a traceback.</i>	Đọc dòng cuối của traceback để biết loại lỗi, dòng trên để biết vị trí. <i>Read the last line of the traceback for the kind of error, the lines above for the place.</i>
Lỗi logic <i>Logic error</i>	Không tự phát hiện được. <i>Not detected automatically.</i>	Chương trình chạy xong nhưng kết quả sai. <i>The program finishes but the result is wrong.</i>	So kết quả với số tính tay, thêm lệnh print để xem biến trung gian. <i>Compare with a hand calculation, add print calls to see intermediate variables.</i>

Lỗi cú pháp — *Syntax errors*

Thiếu dấu ngoặc đóng ở dòng 2 làm Python không đọc nổi chương trình. Thông báo chỉ ra dấu ngoặc mở chưa được đóng:

A missing closing bracket on line 2 stops Python from reading the program. The message points at the bracket that was never closed:

```
tong_thu = 7_600_000
print("Tổng thu:", tong_thu
print("Xong")
```

Kết quả chạy / *Output*

```
File "chuong_trinh.py", line 2
    print("Tổng thu:", tong_thu
    ^
SyntaxError: '(' was never closed
```

Lỗi lúc chạy — *Runtime errors*

Cú pháp đúng nên chương trình bắt đầu chạy, nhưng phép chia cho 0 làm nó dừng ở dòng 3:

The syntax is correct, so the program starts, but division by zero stops it at line 3:

```
tong_chi = 3_134_000
so_ngay = 0
print(tong_chi / so_ngay)
```

Kết quả chạy / Output

```
Traceback (most recent call last):
  File "chuong_trinh.py", line 3, in <module>
    print(tong_chi / so_ngay)
          ~~~~~^~~~~~
ZeroDivisionError: division by zero
```

Lỗi logic — Logic errors

Dòng 4 cộng tổng chi thay vì trừ. Python không báo gì vì phép cộng hợp lệ; chỉ người đọc kết quả mới phát hiện số dư sai:

Line 4 adds total spending instead of subtracting it. Python reports nothing because addition is valid; only a reader who checks the result notices the wrong balance:

```
SO_DU_DAU_THANG = 2_500_000
tong_thu = 7_600_000
tong_chi = 3_134_000
so_du_cuoi_thang = SO_DU_DAU_THANG + tong_thu + tong_chi
print("Số dư cuối tháng:", so_du_cuoi_thang)
```

Kết quả chạy / Output

```
Số dư cuối tháng: 13234000
```

Kết quả 13234000 lớn hơn cả tổng số tiền có trong tháng, trong khi số tính tay ở mục 1.1 là 6966000. Sửa dấu "+" thành "-" ở dòng 4 sẽ ra đúng.

The result 13234000 is larger than all the money available in the month, while the hand-calculated figure in section 1.1 is 6966000. Changing "+" to "-" on line 4 gives the right value.

Đọc thông báo lỗi — Reading an error message

Khi lỗi lúc chạy xảy ra, Python in **traceback**: các dòng cho biết lỗi nằm ở tệp nào, dòng nào, lệnh nào, rồi dòng cuối ghi tên loại lỗi kèm lời giải thích. Đọc từ dưới lên:

*When a runtime error occurs Python prints a **traceback**: lines that say which file, which line and which statement the error is in, then a last line with the name of the error and an explanation. Read from the bottom up:*

1. Dòng cuối: tên loại lỗi (ví dụ ZeroDivisionError) và lý do (division by zero).
Last line: the name of the error (for example ZeroDivisionError) and the reason (division by zero).
2. Dòng bắt đầu bằng File: số dòng của lệnh gây lỗi trong tệp.
The line starting with File: the line number of the statement that caused the error.
3. Dòng mã được in lại bên dưới: lệnh cụ thể cần xem; kiểm tra giá trị các biến trong lệnh đó.
The source line echoed below it: the exact statement to examine; check the values of the variables in it.

Với lỗi logic không có traceback, cách đơn giản nhất là in các biến trung gian rồi so với số tính tay. Dưới đây in từng bước của phép tính số dư để thấy bước nào sai:

With a logic error there is no traceback, so the simplest way is to print intermediate variables and compare them with the hand calculation. The code below prints each step of the balance calculation to show which step is wrong:

```
SO_DU_DAU_THANG = 2_500_000
tong_thu = 7_600_000
tong_chi = 3_134_000
sau_khi_thu = SO_DU_DAU_THANG + tong_thu
print("Sau khi cộng thu:", sau_khi_thu)
sau_khi_chi = sau_khi_thu + tong_chi
print("Sau khi trừ chi:", sau_khi_chi)
```

Kết quả chạy / Output

```
Sau khi cộng thu: 10100000
Sau khi trừ chi: 13234000
```

Bước đầu cho 10100000, đúng. Bước sau phải nhỏ hơn 10100000 vì đã trừ chi, nhưng lại lớn hơn: lỗi nằm ở dòng cộng chi thay vì trừ.

The first step gives 10100000, which is right. The next step must be smaller than 10100000 because spending was subtracted, yet it is larger: the error is the line that adds spending instead of subtracting.

LỖI HAY GẶP / COMMON MISTAKE — Lỗi logic không có thông báo. Luôn đối chiếu kết quả với một số tính tay trước khi tin chương trình.

A logic error gives no message. Always compare the result with a hand-calculated figure before trusting the program.

1.10. Sổ Xu, bản 0.1 — Sổ Xu, version 0.1

Sổ Xu là chương trình quản lý tài chính cá nhân dòng lệnh do sinh viên Nguyễn Hoàng Minh, năm nhất ngành Công nghệ tài chính ở TP.HCM, viết dần qua từng chương. Số dư đầu tháng 9/2026 là 2.500.000 đồng. Bản 0.1 chỉ in lời chào, tên chủ sổ và số dư cuối tháng; các chương sau thêm nhập giao dịch, rẽ nhánh, vòng lặp, hàm, tệp.

Sổ Xu is a command-line personal finance program written step by step, chapter by chapter, by Nguyễn Hoàng Minh, a first-year Financial Technology student in Ho Chi Minh City. The opening balance for September 2026 is 2,500,000 dong. Version 0.1 only prints a greeting, the owner's name and the end-of-month balance; later chapters add transaction entry, branching, loops, functions and files.

```
TEN_UNG_DUNG = "Sổ Xu"
CHU = "Nguyễn Hoàng Minh"
SO_DU_DAU_THANG = 2_500_000
tong_thu = 7_600_000
tong_chi = 3_134_000

print("=" * 34)
print(TEN_UNG_DUNG, "0.1 - quản lý thu chi cá nhân")
print("=" * 34)
print("Chủ sổ:", CHU)
print("Số dư đầu tháng 9/2026:", SO_DU_DAU_THANG)
print("Tổng thu:", tong_thu)
print("Tổng chi:", tong_chi)
print("Số dư cuối tháng:", SO_DU_DAU_THANG + tong_thu - tong_chi)
```

Kết quả chạy / Output

```
=====
Số Xu 0.1 - quản lý thu chi cá nhân
=====
Chủ sở: Nguyễn Hoàng Minh
Số dư đầu tháng 9/2026: 2500000
Tổng thu: 7600000
Tổng chi: 3134000
Số dư cuối tháng: 6966000
```

Phép nhân chuỗi "=" * 34 lặp dấu bằng 34 lần để kẻ đường ngang. Ba số 7600000, 3134000 và 6966000 khớp với tổng thu, tổng chi và số dư cuối của 12 giao dịch tháng 9/2026 mà các chương sau nhập vào chương trình.

*The string product "=" * 34 repeats the equals sign 34 times to draw a horizontal line. The figures 7600000, 3134000 and 6966000 match the total income, total spending and closing balance of the 12 transactions of September 2026 that later chapters enter into the program.*

Phiên bản 0.1 cố ý chưa dùng input, rẽ nhánh hay vòng lặp. Mỗi số nằm sẵn trong mã nên kết quả luôn là 6966000; việc nhập số từ bàn phím bắt đầu ở các bài tập mục 1.11 và được dạy bài bản ở chương 2.

Version 0.1 deliberately uses no input, branching or loops. Every number sits in the code, so the result is always 6966000; typing numbers at the keyboard starts in the exercises of section 1.11 and is taught properly in chapter 2.

1.11. Bài tập lập trình có lời giải — Programming exercises with solutions

Bài 1. Số dư cuối tháng từ dữ liệu nhập — Exercise 1. End-of-month balance from keyboard input

Đề bài. Viết chương trình Số Xu nhập số dư đầu tháng, tổng thu, tổng chi (đồng) rồi in số dư cuối tháng. Chạy với 2.500.000; 7.600.000; 3.134.000.

Problem. Write a Số Xu program that reads the opening balance, total income and total spending (dong) and prints the end-of-month balance. Run it with 2,500,000; 7,600,000; 3,134,000.

1. Đầu vào: ba số nguyên. Đầu ra: một số nguyên. Công thức: số dư cuối = số dư đầu + thu - chi.

Input: three integers. Output: one integer. Formula: closing = opening + income - spending.

2. "input()" luôn trả về chuỗi nên bọc trong "int()" trước khi tính.

"input()" always returns a string, so wrap it in "int()" before calculating.

```
so_du_dau = int(input("Số dư đầu tháng: "))
tong_thu = int(input("Tổng thu: "))
tong_chi = int(input("Tổng chi: "))
so_du_cuoi_thang = so_du_dau + tong_thu - tong_chi
print("Số dư cuối tháng:", so_du_cuoi_thang)
```

Kết quả chạy / Output

```
Số dư đầu tháng: 2500000
Tổng thu: 7600000
Tổng chi: 3134000
Số dư cuối tháng: 6966000
```

Nhận xét. Quên "int()" thì "so_du_dau + tong_thu" nối hai chuỗi và phép trừ ngay sau đó báo TypeError. Nhập số có dấu chấm như 2.500.000 làm "int()" báo ValueError; phải nhập 2500000.

Remark. Without "int()", "so_du_dau + tong_thu" joins two strings and the subtraction straight after raises TypeError. Typing a number with dots such as 2.500.000 makes "int()" raise ValueError; type 2500000.

Bài 2. Điểm tổng kết học phần — Exercise 2. Course final mark

Đề bài. Dữ kiện đề bài: điểm chuyên cần chiếm 20%, giữa kỳ 30%, cuối kỳ 50%. Một sinh viên có điểm chuyên cần 9, giữa kỳ 7, cuối kỳ 6,5 (thang 10). Viết chương trình nhập ba điểm và in điểm tổng kết làm tròn hai chữ số thập phân.

Problem. Given data: attendance counts for 20%, the midterm for 30% and the final for 50%. A student has 9 for attendance, 7 for the midterm and 6.5 for the final (scale of 10). Write a program that reads the three marks and prints the final mark rounded to two decimal places.

1. Điểm có thể lẻ nên đổi bằng "float()" thay cho "int()". Nhập số thập phân bằng dấu chấm: 6.5.
Marks may be fractional, so convert with "float()" instead of "int()". Type decimals with a point: 6.5.
2. Điểm tổng kết = điểm chuyên cần * 0,2 + giữa kỳ * 0,3 + cuối kỳ * 0,5; trong mã viết 0.2, 0.3, 0.5.
*Final mark = attendance * 0.2 + midterm * 0.3 + final * 0.5.*
3. Dùng "round(x, 2)" để làm tròn khi in.
Use "round(x, 2)" to round when printing.

```
diem_cc = float(input("Điểm chuyên cần: "))
diem_gk = float(input("Điểm giữa kỳ: "))
diem_ck = float(input("Điểm cuối kỳ: "))
diem_tb = diem_cc * 0.2 + diem_gk * 0.3 + diem_ck * 0.5
print("Điểm tổng kết:", round(diem_tb, 2))
```

Kết quả chạy / Output

```
Điểm chuyên cần: 9
Điểm giữa kỳ: 7
Điểm cuối kỳ: 6.5
Điểm tổng kết: 7.15
```

Kết quả 7.15: 1,8 + 2,1 + 3,25. **Nhận xét.** Hệ số viết dạng phần trăm (20) mà quên chia 100 cho điểm tổng kết lớn hơn thang 10; luôn đổi 20% thành 0.2.

The result is 7.15: 1.8 + 2.1 + 3.25. Remark. Writing the weights as percentages (20) and forgetting to divide by 100 gives a mark above the scale of 10; always turn 20% into 0.2.

Bài 3. Chi ăn uống so với ngân sách — Exercise 3. Food spending against the budget

Đề bài. Ngân sách ăn uống của Sở Xu là 1.500.000 đồng. Nhập ba khoản chi ăn uống của tháng 9/2026 (45.000; 230.000; 310.000), in tổng và cho biết còn lại bao nhiêu hoặc vượt bao nhiêu.

Problem. The Sở Xu food budget is 1,500,000 dong. Enter the three food entries of September 2026 (45,000; 230,000; 310,000), print the total and say how much is left or how much is over.

1. Hạn mức lấy từ "NGAN_SACH["ăn uống"]" (từ điển, học ở chương 8; ở đây chỉ cần hiểu là tra hạn mức theo tên danh mục).

The limit comes from "NGAN_SACH["ăn uống"]" (a dictionary, learned in chapter 8; here it is enough to see it as looking up the limit by category name).

2. Tính "con_lai = hạn mức - tổng". Nếu "con_lai" không âm thì còn trong ngân sách; ngược lại in số vượt là "-con_lai".

Compute "con_lai = limit - total". If "con_lai" is not negative the entry is within budget; otherwise print the excess as "-con_lai".

```
NGAN_SACH = {"ăn uống": 1_500_000}
lan_1 = int(input("Khoản chi ăn uống lần 1: "))
lan_2 = int(input("Khoản chi ăn uống lần 2: "))
lan_3 = int(input("Khoản chi ăn uống lần 3: "))
tong_an_uong = lan_1 + lan_2 + lan_3
con_lai = NGAN_SACH["ăn uống"] - tong_an_uong
print("Tổng ăn uống:", tong_an_uong)
if con_lai >= 0:
    print("Trong ngân sách, còn lại:", con_lai)
else:
    print("Vượt ngân sách:", -con_lai)
```

Kết quả chạy / Output

```
Khoản chi ăn uống lần 1: 45000
Khoản chi ăn uống lần 2: 230000
Khoản chi ăn uống lần 3: 310000
Tổng ăn uống: 585000
Trong ngân sách, còn lại: 915000
```

Tổng 585.000 đồng khớp với chi ăn uống của Sổ Xu; còn lại 915.000 đồng. **Nhận xét.** Dùng ">" thay cho ">=" ở điều kiện làm trường hợp chi đúng bằng ngân sách bị báo là vượt; luôn thử cả giá trị biên.

*The total of 585,000 dong matches Sổ Xu food spending; 915,000 dong is left. **Remark.** Using ">" instead of ">=" in the condition reports spending exactly equal to the budget as an excess; always test the boundary value too.*

Bài 4. Sửa lỗi từng bước — Exercise 4. Fixing errors one at a time

Đề bài. Dữ kiện đề bài: gửi tiết kiệm 1.000.000 đồng, lãi 4,8%/năm. Chương trình sau cần in lãi sinh ra trong tháng đầu (lãi tháng bằng lãi năm chia 12) nhưng có ba lỗi, mỗi loại một lỗi. Tìm và sửa từng lỗi.

Problem. Given data: a savings deposit of 1,000,000 dong at 4.8% a year. The following program should print the interest earned in the first month (monthly interest is the annual interest divided by 12) but contains three errors, one of each kind. Find and fix them one at a time.

1. Chạy bản gốc. Python dừng ở dòng 4 với lỗi cú pháp: chuỗi thiếu dấu nháy đóng. Sửa bằng cách thêm dấu nháy.

Run the original. Python stops at line 4 with a syntax error: the string lacks its closing quote. Fix it by adding the quote.

```
so_tien = 1_000_000
lai_suat_nam = 0.048
lai_thang_dau = so_tien * lai_suat_nam
print("Lãi tháng đầu:, lai_thang_dau)
```

Kết quả chạy / Output

```
File "chuong_trinh.py", line 4
    print("Lãi tháng đầu:", lai_thang_dau)
      ^
SyntaxError: unterminated string literal (detected at line 4)
```

2. Sau khi thêm dấu nháy, tên biến trong "print" viết sai thành "lai_thang". Python báo NameError và gợi ý tên đúng.

With the quote added, the variable in "print" is misspelt "lai_thang". Python raises NameError and suggests the right name.

```
so_tien = 1_000_000
lai_suat_nam = 0.048
lai_thang_dau = so_tien * lai_suat_nam
print("Lãi tháng đầu:", lai_thang)
```

Kết quả chạy / Output

```
Traceback (most recent call last):
  File "chuong_trinh.py", line 4, in <module>
    print("Lãi tháng đầu:", lai_thang)
      ^^^^^^^^^^
NameError: name 'lai_thang' is not defined. Did you mean: 'lai_thang_dau'?
```

3. Sửa tên biến thì chương trình chạy nhưng in lãi cả năm vì quên chia 12: lỗi logic.

With the name fixed the program runs but prints a full year of interest because the division by 12 is missing: a logic error.

```
so_tien = 1_000_000
lai_suat_nam = 0.048
lai_thang_dau = so_tien * lai_suat_nam
print("Lãi tháng đầu:", lai_thang_dau)
```

Kết quả chạy / Output

```
Lãi tháng đầu: 48000.0
```

4. Thêm "/" 12" vào dòng 3. Kết quả đúng:

Add "/" 12" to line 3. The correct result:

```
so_tien = 1_000_000
lai_suat_nam = 0.048
lai_thang_dau = so_tien * lai_suat_nam / 12
print("Lãi tháng đầu:", lai_thang_dau)
```

Kết quả chạy / Output

```
Lãi tháng đầu: 4000.0
```

Bản lỗi in 48000.0, bản đúng in 4000.0 đồng, tức 0,4% của 1.000.000, khớp lãi suất tháng của Sổ Xu. **Nhận xét.** Một lần chạy chỉ lộ một lỗi; sửa xong lỗi này mới thấy lỗi kế tiếp. Lỗi cú pháp và lỗi tên lộ ngay, còn lỗi logic chỉ lộ khi so với số tính tay.

*The faulty version prints 48000.0, the correct one 4000.0 dong, which is 0.4% of 1,000,000 and matches the Sổ Xu monthly rate. **Remark.** One run shows only one error; the next appears after this one is fixed. Syntax and name errors show at once, while a logic error shows only against a hand-calculated figure.*

1.12. Lỗi hay gặp khi bắt đầu với Python — *Common mistakes when starting with Python*

LỖI HAY GẶP / COMMON MISTAKE — Viết hoa tên lệnh. Python phân biệt chữ hoa và chữ thường, "Print" không phải "print".
Capitalising a command. Python is case-sensitive, and "Print" is not "print".

```
Print("Xin chào")
```

Kết quả chạy / Output

```
Traceback (most recent call last):
  File "chuong_trinh.py", line 1, in <module>
    Print("Xin chào")
    ^^^^^
NameError: name 'Print' is not defined. Did you mean: 'print'?
```

LỖI HAY GẶP / COMMON MISTAKE — Dán mã từ Word hoặc trang web có dấu nhảy cong. Python chỉ nhận dấu nhảy thẳng " hoặc '.
Pasting code from Word or a web page with curly quotes. Python accepts only straight quotes " or '.

```
print("Xin chào")
```

Kết quả chạy / Output

```
File "chuong_trinh.py", line 1
  print("Xin chào")
      ^
SyntaxError: invalid character '"' (U+201C)
```

LỖI HAY GẶP / COMMON MISTAKE — Gõ lệnh của terminal vào trong REPL. Dòng "python3 chao.py" chỉ gõ ở terminal; gõ sau dấu nhắc ">>>" thì Python đọc nó như mã Python.
Typing a terminal command inside the REPL. The line "python3 chao.py" belongs at the terminal; typed after the ">>>" prompt it is read as Python code.

```
python3 chao.py
```

Kết quả chạy / Output

```
File "chuong_trinh.py", line 1
  python3 chao.py
      ^^^^^
SyntaxError: invalid syntax
```

LỖI HAY GẶP / COMMON MISTAKE — Thụt lề các dòng cùng khối không đều nhau, hoặc trộn tab với dấu cách. Dùng bốn dấu cách và để VS Code đổi phím Tab thành dấu cách.
Indenting the lines of one block unevenly, or mixing tabs and spaces. Use four spaces and let VS Code turn the Tab key into spaces.

LỖI HAY GẶP / COMMON MISTAKE — Đặt tên tệp trùng tên module chuẩn, ví dụ "random.py" hoặc "math.py". Lệnh "import" sau đó nạp tệp của chính mình thay cho module chuẩn.
Naming a file after a standard module, such as "random.py" or "math.py". A later "import" then loads your own file instead of the standard module.

LỖI HAY GẶP / COMMON MISTAKE — Chạy lệnh ở sai thư mục. Lệnh "python3 chao.py" cần terminal đang ở thư mục chứa tệp; kiểm tra thư mục hiện tại trước khi chạy.
Running the command in the wrong folder. The command "python3 chao.py" needs the terminal to be in the folder that holds the file; check the current folder first.

1.13. Bảng thuật ngữ — Glossary

Thuật ngữ <i>Term (Vietnamese)</i>	Tiếng Anh <i>English</i>	Giải thích ngắn <i>Short definition</i>
thuật toán	algorithm	Dãy hữu hạn các bước xác định để giải một bài toán. <i>A finite sequence of definite steps that solves a problem.</i>
chương trình	program	Thuật toán viết bằng ngôn ngữ lập trình. <i>An algorithm written in a programming language.</i>
tính hữu hạn	finiteness	Thuật toán dừng sau số bước hữu hạn. <i>The algorithm stops after finitely many steps.</i>
tính xác định	definiteness	Mỗi bước không mơ hồ. <i>Every step is unambiguous.</i>
đầu vào, đầu ra	input, output	Dữ liệu đưa vào và kết quả đưa ra. <i>The data supplied and the result produced.</i>
lưu đồ	flowchart	Sơ đồ ký hiệu và mũi tên biểu diễn thuật toán. <i>A diagram of symbols and arrows that shows an algorithm.</i>
mã giả	pseudocode	Các bước viết bằng câu lệnh ngắn, không theo cú pháp ngôn ngữ nào. <i>Steps written as short statements that follow no language's syntax.</i>
ngôn ngữ máy	machine language	Mã nhị phân bộ xử lý thực hiện trực tiếp. <i>Binary code the processor runs directly.</i>
hợp ngữ	assembly language	Ngôn ngữ dùng tên gọi nhớ cho từng lệnh máy. <i>A language with mnemonic names for single machine instructions.</i>
ngôn ngữ bậc cao	high-level language	Ngôn ngữ gần con người, không phụ thuộc loại máy. <i>A language close to people, independent of the machine type.</i>
biên dịch	compile	Dịch toàn bộ chương trình thành mã máy trước khi chạy. <i>Translate the whole program to machine code before running.</i>

Thuật ngữ <i>Term (Vietnamese)</i>	Tiếng Anh <i>English</i>	Giải thích ngắn <i>Short definition</i>
thông dịch	interpret	Dịch và thực hiện chương trình mỗi lần chạy. <i>Translate and execute the program at each run.</i>
bytecode	bytecode	Mã trung gian Python biên dịch ra để máy ảo chạy. <i>The intermediate code Python compiles to for its virtual machine.</i>
trình thông dịch	interpreter	Phần mềm đọc và thực hiện mã nguồn Python. <i>The software that reads and executes Python source code.</i>
REPL	REPL	Chế độ tương tác: đọc, tính, in, lặp. <i>Interactive mode: read, evaluate, print, loop.</i>
workspace	workspace	Thư mục (hoặc nhóm thư mục) mở trong VS Code, tương ứng solution. <i>The folder (or group of folders) open in VS Code, matching a solution.</i>
terminal tích hợp	integrated terminal	Cửa sổ dòng lệnh nằm trong VS Code. <i>The command-line window inside VS Code.</i>
chú thích	comment	Phần sau dấu # mà Python bỏ qua. <i>Text after # that Python ignores.</i>
thụt lề	indentation	Khoảng trắng đầu dòng xác định khối lệnh. <i>Leading spaces that define a block.</i>
PEP 8	PEP 8	Hướng dẫn phong cách viết mã Python. <i>The style guide for Python code.</i>
lỗi cú pháp	syntax error	Mã vi phạm quy tắc viết, Python không đọc được. <i>Code that breaks the rules of writing, so Python cannot read it.</i>
lỗi lúc chạy	runtime error	Lỗi xảy ra khi đang chạy, làm chương trình dừng. <i>An error during running that stops the program.</i>
lỗi logic	logic error	Chương trình chạy nhưng cho kết quả sai. <i>The program runs but gives a wrong result.</i>

1.14. Câu hỏi tự luận và dàn ý trả lời — *Essay questions and answer outlines*

Câu 1. Trình bày khái niệm thuật toán và năm tính chất của thuật toán. Minh họa bằng thuật toán tính số dư cuối tháng của Sổ Xu.

Question 1. *Explain what an algorithm is and its five properties. Illustrate with the Sổ Xu end-of-month balance algorithm.*

- Định nghĩa thuật toán và phân biệt thuật toán với chương trình.
Define an algorithm and distinguish it from a program.
- Nêu và giải thích năm tính chất: hữu hạn, xác định, đúng, có đầu vào và đầu ra, hiệu quả.
State and explain the five properties: finiteness, definiteness, correctness, input and output, effectiveness.
- Viết thuật toán bằng ngôn ngữ tự nhiên: lấy số dư đầu tháng 2.500.000, cộng tổng thu 7.600.000, trừ tổng chi 3.134.000.
Write the algorithm in natural language: take the opening balance 2,500,000, add total income 7,600,000, subtract total spending 3,134,000.
- Chỉ ra đầu vào, đầu ra và kết quả 6.966.000 đồng.
Identify the input, the output and the result 6,966,000 dong.
- Giải thích vì sao thuật toán này hữu hạn và xác định.
Explain why this algorithm is finite and definite.

Câu 2. So sánh ba cách biểu diễn thuật toán và so sánh biên dịch với thông dịch; cho biết Python thuộc nhóm nào.

Question 2. *Compare the three ways of representing an algorithm, compare compiling with interpreting, and say which group Python belongs to.*

- Ưu điểm và hạn chế của ngôn ngữ tự nhiên, lưu đồ, mã giả.
Strengths and weaknesses of natural language, flowcharts and pseudocode.
- Ý nghĩa bốn ký hiệu lưu đồ: bắt đầu và kết thúc, nhập và xuất, xử lý, điều kiện.
The meaning of four flowchart symbols: start and end, input and output, processing, decision.
- Khác nhau giữa biên dịch và thông dịch về thời điểm dịch, sản phẩm, phát hiện lỗi.
How compiling and interpreting differ in when translation happens, the product and finding errors.
- Quá trình Python biên dịch sang bytecode rồi máy ảo chạy bytecode; vì sao vẫn gọi là ngôn ngữ thông dịch.
Python compiling to bytecode and the virtual machine running it; why it is still called interpreted.
- Bốn đặc điểm của Python: cú pháp ngắn, thụt lề, kiểu động, thư viện chuẩn.
Four features of Python: short syntax, indentation, dynamic typing, the standard library.

Câu 3. Trình bày ba loại lỗi trong lập trình và cách đưa một chương trình Python chạy trong VS Code.

Question 3. *Describe the three kinds of programming error and how to run a Python program in VS Code.*

- Định nghĩa lỗi cú pháp, lỗi lúc chạy, lỗi logic và thời điểm phát hiện mỗi loại.
Define syntax, runtime and logic errors and when each is found.
- Cho một ví dụ mã cho từng loại: thiếu ngoặc đóng, chia cho 0, cộng thay vì trừ khi tính số dư.
Give a code example for each: a missing closing bracket, division by zero, adding instead of subtracting in the balance.
- Cách đọc traceback: dòng cuối cho loại lỗi, các dòng trên cho vị trí.
How to read a traceback: the last line gives the kind of error, the lines above give the place.
- Các bước chuẩn bị: cài Python, cài VS Code và phần mở rộng Python, chọn trình thông dịch, mở thư mục làm việc.
The preparation steps: install Python, install VS Code and the Python extension, select the interpreter, open the working folder.
- Hai cách chạy: tệp .py trong terminal tích hợp và REPL; nêu khi nào dùng cách nào.
Two ways to run: a .py file in the integrated terminal and the REPL; when to use each.
- Tương ứng solution và project với workspace và thư mục dự án.
How solution and project correspond to workspace and project folder.

1.15. Câu hỏi trắc nghiệm — Multiple-choice questions

1. Tính chất nào sau đây KHÔNG thuộc năm tính chất của thuật toán?

Which of the following is NOT one of the five properties of an algorithm?

- A. Tính hữu hạn / *Finiteness*
- B. Tính xác định / *Definiteness*
- C. Tính đúng / *Correctness*
- D. Tính ngẫu nhiên: mỗi lần chạy có thể cho kết quả khác nhau / *Randomness: each run may give a different result*

2. Trong lưu đồ, hình thoi biểu diễn gì?

In a flowchart, what does a diamond represent?

- A. Bắt đầu hoặc kết thúc / *Start or end*
- B. Điều kiện, rẽ nhánh / *A condition, a branch*
- C. Nhập hoặc xuất dữ liệu / *Input or output*
- D. Phép gán, xử lý / *Assignment, processing*

3. Lệnh "a, b = 48, 18; a, b = b, a % b; print(a, b)" in ra gì?

What does the statement "a, b = 48, 18; a, b = b, a % b; print(a, b)" print?

- A. 18 12
- B. 48 18
- C. 12 18
- D. 18 30

4. Phát biểu nào đúng về Python?

Which statement about Python is correct?

- A.** Là ngôn ngữ máy, chạy trực tiếp trên bộ xử lý / *It is a machine language that runs directly on the processor*
- B.** Là ngôn ngữ bậc cao, mã nguồn được trình thông dịch Python thực hiện / *It is a high-level language whose source is executed by the Python interpreter*
- C.** Là hợp ngữ, mỗi lệnh ứng với một lệnh máy / *It is an assembly language, each statement matching one machine instruction*
- D.** Chỉ dùng để thiết kế trang web / *It is used only to design web pages*

5. Lệnh "# print('A'); print('B')" in ra gì?

What does the line "# print('A'); print('B')" print?

- A.** A rồi B, hai dòng / *A then B, on two lines*
- B.** Chỉ B / *Only B*
- C.** Không in gì / *Nothing*
- D.** Báo SyntaxError / *A SyntaxError*

6. Chương trình tính số dư cuối tháng chạy xong, không báo lỗi, nhưng in 13234000 thay cho 6966000 vì dùng "+" thay cho "-" khi trừ tổng chi. Đây là loại lỗi nào?

A program for the end-of-month balance finishes with no error message but prints 13234000 instead of 6966000 because it uses "+" instead of "-" for the spending. What kind of error is this?

- A.** Lỗi logic / *Logic error*
- B.** Lỗi cú pháp / *Syntax error*
- C.** Lỗi lúc chạy / *Runtime error*
- D.** Lỗi thụt lề / *Indentation error*

7. Lệnh "so_du = 100; so_du = so_du - 30; print(so_du)" in ra gì?

What does the statement "so_du = 100; so_du = so_du - 30; print(so_du)" print?

- A.** 100
- B.** 30
- C.** 130
- D.** 70

8. Theo PEP 8, tên nào sau đây hợp lệ và đúng quy ước đặt tên cho một biến?

According to PEP 8, which of the following is a valid, conventional name for a variable?

- A.** SoDuCuoi
- B.** soDuCuoi
- C.** so_du_cuoi
- D.** so du cuoi

Chương 2. Biến, hằng, kiểu dữ liệu và biểu thức — Chapter 2.

Variables, constants, data types and expressions

Mục tiêu chương — Chapter objectives

- Tạo biến bằng phép gán, gán nhiều giá trị, hoán đổi hai biến và đặt tên đúng quy tắc, đúng PEP 8.
Create variables with assignment, assign several values at once, swap two variables and name them by the rules and by PEP 8.
- Phân biệt int, float, str, bool, None; dùng type() và isinstance(); biết vì sao 0.1 + 0.2 khác 0.3 và khi nào dùng Decimal.
Tell int, float, str, bool and None apart; use type() and isinstance(); explain why 0.1 + 0.2 differs from 0.3 and when to use Decimal.
- Viết và tính biểu thức số học với // % **, số âm, thứ tự ưu tiên và phép gán kết hợp.
*Write and evaluate arithmetic expressions with // % **, negative numbers, precedence and augmented assignment.*
- Nhập dữ liệu bằng input(), ép kiểu, xử lý ValueError, xuất bằng print() và f-string, đổi số sang kiểu Việt Nam.
Read data with input(), convert types, recognise ValueError, print with print() and f-strings, and switch numbers to Vietnamese style.
- Xử lý chuỗi cơ bản (nối, lặp, chỉ số, cắt lát, upper, strip, split) và viết bản 0.2 của Sổ Xu.
Handle basic strings (concatenation, repetition, indexing, slicing, upper, strip, split) and write version 0.2 of Sổ Xu.

2.1. Biến và phép gán — Variables and assignment

Biến (variable) là một tên gắn với một đối tượng nằm trong bộ nhớ. Lệnh gán "tên = biểu thức" tính biểu thức ở vế phải trước, rồi gắn tên ở vế trái với kết quả. Python không cần khai báo kiểu: tên xuất hiện lần đầu ở vế trái của một phép gán là biến được tạo ra. Gán lại cùng tên thì tên đó chuyển sang gắn với đối tượng mới.

*A **variable** is a name attached to an object held in memory. The assignment statement "name = expression" evaluates the right-hand side first and then binds the name on the left to the result. Python needs no type declaration: a name that appears for the first time on the left of an assignment creates the variable. Assigning to the same name again re-binds it to a new object.*

```
so_du = 2_500_000
print(so_du)
print(type(so_du))
so_du = so_du + 4_000_000
print(so_du)
```

Kết quả chạy / Output

```
2500000
<class 'int'>
6500000
```

Dòng 4 tính $2.500.000 + 4.000.000$ ở vế phải, rồi gán lại tên "so_du" với kết quả 6.500.000. Dấu gạch dưới trong số (2_500_000) chỉ giúp mắt đọc, không ảnh hưởng giá trị.

Line 4 evaluates $2,500,000 + 4,000,000$ on the right and re-binds the name "so_du" to 6,500,000. The underscores inside the number (2_500_000) only help the eye and do not change the value.

PHẢI THUỘC / MEMORISE — Phép gán: ten = biểu_thức. Vế phải tính trước. Biến là tên gán với đối tượng, không phải ô nhớ chứa giá trị.

Assignment: name = expression. The right side is evaluated first. A variable is a name bound to an object, not a box holding a value.

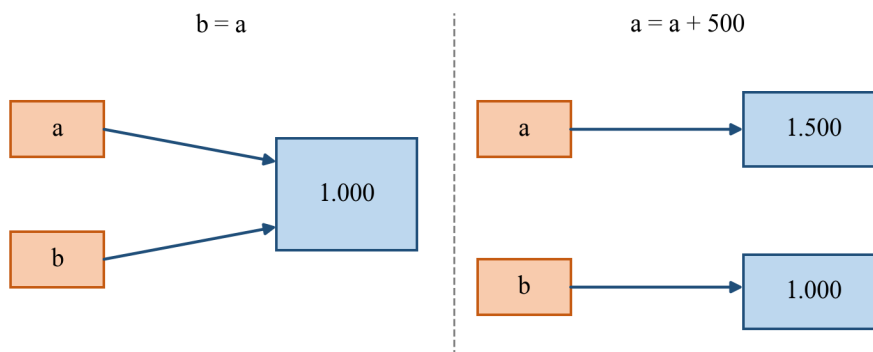
```
a = 1_000
b = a
print(a is b)
a = a + 500
print(a, b, a is b)
```

Kết quả chạy / Output

```
True
1500 1000 False
```

Sau "b = a", hai tên cùng gán với một đối tượng (kết quả "True" của "is"). Phép "a = a + 500" tạo đối tượng mới 1500 và chỉ chuyển tên "a" sang đó; "b" vẫn gán với 1000.

After "b = a" both names are bound to one object ("is" gives "True"). The statement "a = a + 500" creates a new object 1500 and moves only the name "a" to it; "b" still refers to 1000.



Hình 2.1. Hai tên cùng gán một đối tượng; gán lại "a" chỉ chuyển tên "a" sang đối tượng mới

Figure 2.1. Two names bound to one object; re-assigning "a" moves only the name "a" to a new object

Ký hiệu Symbol	Vai trò Role	Ví dụ Example
=	Gán: gán tên với giá trị <i>Assignment: bind a name to a value</i>	so_du = 100
==	So sánh bằng, cho True hoặc False <i>Equality test, gives True or False</i>	so_du == 100

Gán nhiều cho phép khai báo vài biến trên một dòng. Vế phải được gom thành một bộ giá trị rồi mới gán, vì vậy **hoán đổi** hai biến không cần biến trung gian.

*Multiple assignment creates several variables on one line. The right side is first packed into a tuple and only then assigned, so **swapping** two variables needs no temporary variable.*

```
ho, ten = "Nguyễn", "Minh"
x = y = z = 0
thu, chi = 7_600_000, 3_134_000
print(ho, ten, x, y, z)
print(thu - chi)
```

Kết quả chạy / Output

```
Nguyễn Minh 0 0 0
4466000
```

```
a, b = 5, 9
tam = a
a = b
b = tam
print(a, b)
a, b = b, a
print(a, b)
```

Kết quả chạy / Output

```
9 5
5 9
```

Cách thứ nhất dùng biến trung gian "tam" để không mất giá trị của "a". Cách thứ hai "a, b = b, a" đổi chỗ trong một lệnh và là cách viết thường dùng trong Python.

The first way uses a temporary variable "tam" so the value of "a" is not lost. The second, "a, b = b, a", swaps in one statement and is the usual Python idiom.

LỖI HAY GẶP / COMMON MISTAKE — Số biến bên trái phải bằng số giá trị bên phải.

Lệnh "a, b = 1, 2, 3" báo ValueError: too many values to unpack.

The number of names on the left must equal the number of values on the right. "a, b = 1, 2, 3" raises ValueError: too many values to unpack.

```
a, b = 1, 2, 3
```

Kết quả chạy / Output

```
Traceback (most recent call last):
  File "chuong_trinh.py", line 1, in <module>
    a, b = 1, 2, 3
    ^^^
ValueError: too many values to unpack (expected 2)
```

2.2. Quy tắc đặt tên, từ khóa và hằng — Naming rules, keywords and constants

Quy tắc <i>Rule</i>	Đúng <i>Valid</i>	Sai <i>Invalid</i>
Gồm chữ cái, chữ số, dấu gạch dưới <i>Letters, digits and underscores</i>	so_du, tong_chi2	so du, tong-chi
Không bắt đầu bằng chữ số <i>Must not start with a digit</i>	thang9	9thang

Quy tắc <i>Rule</i>	Đúng <i>Valid</i>	Sai <i>Invalid</i>
Phân biệt chữ hoa, chữ thường <i>Case-sensitive</i>	so_du và So_du là hai tên	—
Không trùng từ khóa <i>Must not be a keyword</i>	loai_gd	class, if, for

Python 3 cho phép chữ có dấu trong tên, nhưng quy ước PEP 8 và ứng dụng Sổ Xu dùng tên viết thường không dấu, ngăn từ bằng dấu gạch dưới (kiểu snake_case): "so_du", "tong_chi". Tên nên mô tả đúng nội dung biến.

Python 3 allows accented letters in names, but PEP 8 and the Sổ Xu application use lower-case ASCII names with underscores between words (snake_case): "so_du", "tong_chi". A name should describe what the variable holds.

```
so_du = 100
So_du = 200
print(so_du, So_du)
print(so_du == So_du)
```

Kết quả chạy / Output

```
100 200
False
```

```
import keyword
print(keyword.iskeyword("if"))
print(keyword.iskeyword("so_du"))
print(keyword.iskeyword("print"))
```

Kết quả chạy / Output

```
True
False
False
```

Từ khóa (keyword) là từ Python giữ riêng cho cú pháp, như "if", "for", "class", "True", "None". "print" không phải từ khóa mà là tên hàm dựng sẵn; vì vậy gán đè lên nó là hợp lệ về cú pháp và gây lỗi ở lệnh sau.

*A **keyword** is a word Python reserves for its syntax, such as "if", "for", "class", "True" and "None". "print" is not a keyword but a built-in function name, so assigning over it is legal syntax and causes an error in a later statement.*

LỖI HAY GẶP / COMMON MISTAKE — Đặt tên biến trùng hàm dựng sẵn (sum, max, len, type, list). Sau khi gán "sum = 3134000", lời gọi sum([...]) báo TypeError: 'int' object is not callable.

Naming a variable after a built-in function (sum, max, len, type, list). After "sum = 3134000", the call sum([...]) raises TypeError: 'int' object is not callable.

```
sum = 3_134_000
tong = sum([1_800_000, 585_000])
```

Kết quả chạy / Output

```
Traceback (most recent call last):
  File "chuong_trinh.py", line 2, in <module>
    tong = sum([1_800_000, 585_000])
            ^^^^^^^^^^^^^^^^^^^^^^^^^
TypeError: 'int' object is not callable
```

LỖI HAY GẶP / COMMON MISTAKE — Tên bắt đầu bằng chữ số hoặc có dấu cách là lỗi cú pháp (SyntaxError); chương trình không chạy dòng nào.

A name that starts with a digit or contains a space is a syntax error (SyntaxError); the program does not run at all.

```
9thang = 6_966_000
```

Kết quả chạy / Output

```
File "chuong_trinh.py", line 1
  9thang = 6_966_000
    ^
SyntaxError: invalid decimal literal
```

Đối tượng <i>Item</i>	Quy ước PEP 8 <i>PEP 8 convention</i>	Ví dụ <i>Example</i>
Biến <i>Variable</i>	chữ thường, ngăn từ bằng gạch dưới <i>lower case, words joined by underscores</i>	tong_chi, so_du_cuoi
Hằng <i>Constant</i>	CHỮ_HOÀ, ngăn từ bằng gạch dưới <i>UPPER_CASE, words joined by underscores</i>	LAI_SUAT_NAM
Hàm <i>Function</i>	như biến <i>like a variable</i>	tinh_lai (chương 5 <i>chapter 5</i>)
Lớp <i>Class</i>	mỗi từ viết hoa chữ đầu <i>each word capitalised</i>	GiaoDich (chương 8 <i>chapter 8</i>)
Dòng mã <i>Code line</i>	tối đa 79 ký tự, thụt lề 4 dấu cách <i>at most 79 characters, 4-space indent</i>	—

Từ khóa <i>Keyword</i>	Dùng để <i>Used for</i>	Học ở <i>Covered in</i>
if, elif, else	rẽ nhánh <i>selection</i>	Chương 3 <i>Chapter 3</i>
for, while, break, continue	lặp <i>repetition</i>	Chương 4 <i>Chapter 4</i>
def, return, lambda	định nghĩa hàm <i>defining functions</i>	Chương 5 <i>Chapter 5</i>
and, or, not, in, is	toán tử logic và kiểm tra <i>logical and test operators</i>	Chương 3 <i>Chapter 3</i>

Từ khóa <i>Keyword</i>	Dùng để <i>Used for</i>	Học ở <i>Covered in</i>
True, False, None	giá trị đặc biệt <i>special values</i>	Chương 2 <i>Chapter 2</i>
import, class, try, except	module, lớp, xử lý lỗi <i>modules, classes, error handling</i>	Chương 5, 8, 9 <i>Chapters 5, 8, 9</i>

```
số_du = 100
print(số_du)
so_du_cuoi_thang = 6_966_000
print(so_du_cuoi_thang)
```

Kết quả chạy / Output

```
100
6966000
```

Tên có dấu như "số_du" chạy được nhưng dễ gõ sai khi bàn phím không có bộ gõ tiếng Việt và gây lỗi khi chia sẻ mã; Số Xu giữ tên không dấu, đủ dài để đọc hiểu ("so_du_cuoi_thang" rõ hơn "sdc").

A name with accents such as "số_du" runs but is easy to mistype without a Vietnamese input method and causes trouble when code is shared; Số Xu keeps unaccented names that are long enough to read ("so_du_cuoi_thang" is clearer than "sdc").

Hằng (constant) là giá trị không đổi trong suốt chương trình. Python không có từ khóa tạo hằng; người viết theo quy ước đặt tên CHỮ_HOÀ và tự giữ không gán lại. Trong Số Xu, các hằng dữ kiện gồm "SO_DU_DAU_THANG", "LAI_SUAT_NAM", "MUC_TIEU".

*A **constant** is a value that does not change while the program runs. Python has no keyword to create one; by convention the name is written in UPPER_CASE and the programmer simply never re-assigns it. In Số Xu the constants include "SO_DU_DAU_THANG", "LAI_SUAT_NAM" and "MUC_TIEU".*

```
LAI_SUAT_NAM = 0.048
SO_THANG = 12
lai_thang = LAI_SUAT_NAM / SO_THANG
print(lai_thang)
print(f"{lai_thang:.1%}")
```

Kết quả chạy / Output

```
0.004
0.4%
```

PHẢI THUỘC / MEMORISE — Biến: chữ thường, snake_case (so_du). Hằng: CHỮ_HOÀ (LAI_SUAT_NAM). Tên không trùng từ khóa và hàm dựng sẵn.

Variables: lower-case snake_case (so_du). Constants: UPPER_CASE (LAI_SUAT_NAM). Never reuse a keyword or built-in name.

VÍ DỤ / EXAMPLE — Tính lãi tháng của khoản tiết kiệm: lai_thang = LAI_SUAT_NAM / 12 để đọc hơn x = 0.048 / 12, vì tên cho biết giá trị đại diện cho điều gì.

Monthly interest on savings: lai_thang = LAI_SUAT_NAM / 12 reads better than $x = 0.048 / 12$, because the name says what the value represents.

2.3. Kiểu dữ liệu cơ bản — Basic data types

Kiểu Type	Ý nghĩa Meaning	Ví dụ Example	Khả biến Mutable
int	Số nguyên <i>Integer</i>	12, -5, 2_500_000	Không <i>No</i>
float	Số thực dấu phẩy động <i>Floating-point number</i>	0.004, 3.14	Không <i>No</i>
str	Chuỗi ký tự <i>Character string</i>	"Sổ Xu", 'chi'	Không <i>No</i>
bool	Đúng hoặc sai <i>True or false</i>	True, False	Không <i>No</i>
NoneType	Không có giá trị <i>No value</i>	None	Không <i>No</i>

```
so_giao_dich = 12
ty_le = 0.004
ten = "Sổ Xu"
da_vuot = True
ghi_chu = None
print(type(so_giao_dich).__name__, so_giao_dich)
print(type(ty_le).__name__, ty_le)
print(type(ten).__name__, ten)
print(type(da_vuot).__name__, da_vuot)
print(type(ghi_chu).__name__, ghi_chu)
```

Kết quả chạy / Output

```
int 12
float 0.004
str Sổ Xu
bool True
NoneType None
```

Python là ngôn ngữ **kiểu động** (dynamic typing): kiểu gắn với giá trị chứ không gắn với tên, nên một tên có thể lần lượt gắn với các kiểu khác nhau.

*Python is **dynamically typed**: the type belongs to the value, not to the name, so one name can be bound in turn to values of different types.*

```
x = 10
print(type(x))
x = "mười"
print(type(x))
x = 2.5
print(type(x))
```

Kết quả chạy / Output

```
<class 'int'>
<class 'str'>
<class 'float'>
```

int không bị tràn số: Python cấp thêm bộ nhớ khi số lớn lên, nên chỉ bị giới hạn bởi bộ nhớ máy. Kiểu **float** có độ chính xác hữu hạn (mục 2.5).

*An **int** never overflows: Python allocates more memory as the number grows, so it is limited only by machine memory. A **float** has finite precision (section 2.5).*

```
lon = 2 ** 100
print(lon)
print(type(lon).__name__)
print(len(str(2 ** 1000)))
```

Kết quả chạy / Output

```
1267650600228229401496703205376
int
302
```

"type()" cho biết kiểu chính xác của một giá trị. "isinstance(x, kiểu)" kiểm tra x có thuộc kiểu đó (hoặc một trong các kiểu trong bộ) hay không và được ưu tiên khi viết điều kiện. "bool" là lớp con của "int" nên "True" được tính là 1.

"type()" gives the exact type of a value. "isinstance(x, type)" tests whether x belongs to that type (or to one of the types in a tuple) and is preferred in conditions. "bool" is a subclass of "int", so "True" counts as 1.

```
x = True
print(isinstance(x, int))
print(isinstance(x, bool))
print(type(x) == int)
print(isinstance(3.0, (int, float)))
print(True + True)
```

Kết quả chạy / Output

```
True
True
False
True
2
```

```
ket_qua = None
print(ket_qua is None)
print(ket_qua == 0)
print(None == "")
print(type(None).__name__)
```

Kết quả chạy / Output

```
True
False
False
NoneType
```

"None" biểu diễn "chưa có giá trị" và khác 0, khác chuỗi rỗng. Kiểm tra bằng "is None", không dùng "== None".

"None" means "no value yet" and differs from 0 and from the empty string. Test it with "is None", not "== None".

Các giá trị "trống" khác nhau về kiểu dù cùng mang nghĩa "không có gì": 0 là số, "" là chuỗi rỗng, None là không có giá trị. Phép "==" so sánh nội dung, không so sánh kiểu: 0 == 0.0 là True nhưng "5" == 5 là False.

Different "empty" values differ in type although all mean "nothing": 0 is a number, "" an empty string, None no value. The "==" operator compares content, not type: 0 == 0.0 is True but "5" == 5 is False.

```
print(None == 0, None == "", 0 == False, 0 == "")
print(0 == 0.0, "5" == 5, 5 == 5.0)
print(type(0.0).__name__, type("").__name__)
```

Kết quả chạy / Output

```
False False True False
True False True
float str
```

PHẢI THUỘC / MEMORISE — `type(x)` cho kiểu của `x`; `isinstance(x, int)` cho True hoặc False. Kiểu cơ bản: `int`, `float`, `str`, `bool`, `NoneType`.

type(x) gives the type of x; isinstance(x, int) gives True or False. Basic types: int, float, str, bool, NoneType.

2.4. Toán tử số học và thứ tự ưu tiên — Arithmetic operators and precedence

Toán tử <i>Operator</i>	Tên <i>Name</i>	Ví dụ <i>Example</i>	Kết quả <i>Result</i>
+	Cộng <i>Addition</i>	17 + 5	22
-	Trừ <i>Subtraction</i>	17 - 5	12
*	Nhân <i>Multiplication</i>	17 * 5	85
/	Chia, luôn cho float <i>Division, always a float</i>	17 / 5	3.4
//	Chia nguyên, làm tròn xuống <i>Floor division</i>	17 // 5	3
%	Chia lấy dư <i>Remainder (modulo)</i>	17 % 5	2
**	Lũy thừa <i>Exponentiation</i>	2 ** 10	1024

```
print(17 + 5, 17 - 5, 17 * 5)
print(17 / 5, 17 // 5, 17 % 5, 2 ** 10)
print(8 / 2)
print(type(8 / 2).__name__)
```

Kết quả chạy / Output

```
22 12 85
3.4 3 2 1024
4.0
float
```

Biểu thức <i>Expression</i>	Dùng khi <i>Use it when</i>	Kiểu kết quả <i>Result type</i>
a / b	cần giá trị thập phân <i>you need a decimal value</i>	float
$a // b$	cần thương nguyên (số tờ tiền, số trang) <i>you need a whole quotient (notes, pages)</i>	int nếu cả hai là int <i>int if both are int</i>
$a \% b$	cần phần dư (số lẻ, kiểm tra chẵn lẻ) <i>you need the remainder (leftovers, parity)</i>	int nếu cả hai là int <i>int if both are int</i>

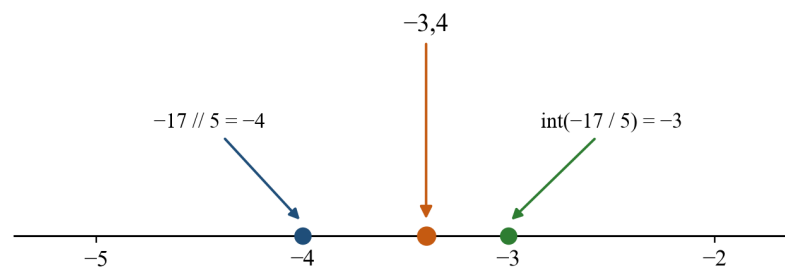
Với số âm, "/" luôn làm tròn **xuống** (về phía $-\infty$) và dấu của "%" theo số chia, sao cho $a == (a // b) * b + a \% b$. Hàm int() thì cắt bỏ phần thập phân (về phía 0), nên "-17 // 5" và "int(-17 / 5)" cho hai kết quả khác nhau.

*With negative numbers, "/" always rounds **down** (towards $-\infty$) and the sign of "%" follows the divisor, so that $a == (a // b) * b + a \% b$. The int() function truncates the fractional part (towards 0), so "-17 // 5" and "int(-17 / 5)" differ.*

```
print(-17 // 5, -17 % 5)
print(17 // -5, 17 % -5)
print(int(-17 / 5))
a, b = -17, 5
print(a == (a // b) * b + a % b)
```

Kết quả chạy / Output

```
-4 3
-4 -3
-3
True
```



Hình 2.2. -17 chia 5 bằng -3,4: "/" lấy -4, int() lấy -3
Figure 2.2. -17 divided by 5 is -3.4: "/" gives -4, int() gives -3

Cách <i>Method</i>	Hướng làm tròn <i>Direction</i>	-3,4 thành <i>-3.4 becomes</i>
// (floor division)	xuống, về $-\infty$ <i>down, towards $-\infty$</i>	-4
int(), math.trunc()	về 0, bỏ phần thập phân <i>towards 0, drops the fraction</i>	-3

Cách <i>Method</i>	Hướng làm tròn <i>Direction</i>	−3,4 thành <i>−3.4 becomes</i>
math.floor()	xuống <i>down</i>	−4
math.ceil()	lên <i>up</i>	−3
round()	về số chẵn gần nhất khi đúng .5 <i>to the nearest even number at exactly .5</i>	−3

```
import math
print(math.floor(-3.4), math.ceil(-3.4))
print(math.trunc(-3.4), int(-3.4), round(-3.4))
print(math.sqrt(144), math.pi)
print(abs(-250_000), max(3, 9, 4), min(3, 9, 4))
print(pow(2, 10), divmod(1_850_000, 500_000))
```

Kết quả chạy / Output

```
-4 -3
-3 -3 -3
12.0 3.141592653589793
250000 9 3
1024 (3, 350000)
```

"divmod(a, b)" trả về cặp (thương nguyên, số dư) trong một lần tính. Ví dụ "divmod(1_850_000, 500_000)" cho biết có 3 tờ 500.000 đồng và còn dư 350.000 đồng.

"divmod(a, b)" returns the pair (whole quotient, remainder) in one step. For example "divmod(1_850_000, 500_000)" shows 3 notes of 500,000 dong with 350,000 dong left over.

Hai ứng dụng thường gặp của "/" và "%": kiểm tra chia hết ("a % b == 0") và tách chữ số ("% 10" lấy chữ số hàng đơn vị, "// 10" bỏ chữ số đó).

Two common uses of "/" and "%": testing divisibility ("a % b == 0") and extracting digits ("% 10" gives the units digit, "// 10" removes it).

```
print(18 % 2 == 0, 7 % 2 == 1, 2_500_000 % 1_000 == 0)
n = 1_234
print(n % 10, n // 10 % 10, n // 1000)
print(n // 10, (n // 10) // 10)
```

Kết quả chạy / Output

```
True True True
4 3 1
123 12
```

Thứ tự ưu tiên — Operator precedence

Mức <i>Level</i>	Toán tử <i>Operators</i>	Kết hợp <i>Associativity</i>
Cao nhất (1) <i>Highest (1)</i>	()	Trong ra ngoài <i>Innermost first</i>
2	**	Phải sang trái <i>Right to left</i>

Mức <i>Level</i>	Toán tử <i>Operators</i>	Kết hợp <i>Associativity</i>
3	+x, -x (dấu đơn ngôi) <i>+x, -x (unary sign)</i>	—
4	* / // %	Trái sang phải <i>Left to right</i>
5	+ -	Trái sang phải <i>Left to right</i>
6	== != < > <= >=	Chương 3 <i>Chapter 3</i>
Thấp nhất (7) <i>Lowest (7)</i>	not, and, or	Chương 3 <i>Chapter 3</i>

```
print(2 + 3 * 4, (2 + 3) * 4)
print(-2 ** 2, (-2) ** 2)
print(2 ** 3 ** 2, (2 ** 3) ** 2)
print(100 - 20 - 10, 100 / 5 / 2)
print(7 + 8 // 3 * 2 ** 2 % 5)
```

Kết quả chạy / Output

```
14 20
-4 4
512 64
70 10.0
10
```

Dòng 2: "*" ưu tiên hơn dấu trừ đơn ngôi nên "-2 ** 2" là $-(2^2) = -4$. Dòng 3: "*" kết hợp từ phải sang trái nên "2 ** 3 ** 2" là $2^9 = 512$. Dòng 5 tính lần lượt: $2 ** 2 = 4$; $8 // 3 = 2$; $2 * 4 = 8$; $8 \% 5 = 3$; $7 + 3 = 10$.

Line 2: "" binds tighter than the unary minus, so "-2 ** 2" is $-(2^2) = -4$. Line 3: "*" associates right to left, so "2 ** 3 ** 2" is $2^9 = 512$. Line 5 goes step by step: $2 ** 2 = 4$; $8 // 3 = 2$; $2 * 4 = 8$; $8 \% 5 = 3$; $7 + 3 = 10$.*

PHẢI THUỘC / MEMORISE — Thứ tự: () rồi ** rồi dấu đơn ngôi rồi * / // % rồi + -.

Cùng mức thì tính từ trái sang phải, riêng ** từ phải sang trái. Nghi ngờ thì thêm ngoặc.

*Order: () then ** then unary sign then * / // % then + -. Equal levels go left to right, except ** (right to left). When in doubt, add parentheses.*

Phép gán kết hợp (augmented assignment) gộp phép tính và phép gán: "x += 5" tương đương "x = x + 5". Có đủ các dạng +=, -=, *=, /=, //=, %=, **=.

*Augmented assignment combines an operation with an assignment: "x += 5" is equivalent to "x = x + 5". All forms exist: +=, -=, *=, /=, //=, %=, **=.*

```
so_du = 2_500_000
so_du += 4_000_000
so_du -= 1_800_000
print(so_du)
so_du *= 2
so_du //= 3
so_du %= 1_000_000
print(so_du)
```

Kết quả chạy / Output

```
4700000
133333
```

Biểu thức hỗn hợp int và float — *Mixed int and float expressions*

Khi một phép tính có cả int và float, Python tự đổi int sang float (ép kiểu ngầm) và kết quả là float. bool cũng được đổi thành 1 hoặc 0 trong phép tính. Số viết dạng khoa học $1e3$ hoặc $1.5e-3$ luôn là float.

When an operation mixes int and float, Python silently converts the int to float (implicit conversion) and the result is a float. A bool also becomes 1 or 0 in arithmetic. A number written in scientific form such as $1e3$ or $1.5e-3$ is always a float.

```
print(3 + 2.5, type(3 + 2.5).__name__)
print(10 // 3.0, 10 % 3.0)
print(True + 2.5)
print(1e3, 1.5e-3, 2e0 == 2)
print(5 / 2 * 2, 5 // 2 * 2)
```

Kết quả chạy / Output

```
5.5 float
3.0 1.0
3.5
1000.0 0.0015 True
5.0 4
```

Dòng 2 cho 3.0 và 1.0: `"/"` và `"%"` với một toán hạng float cho kết quả float. Dòng 5 cho thấy `"5 / 2 * 2"` bằng 5.0 còn `"5 // 2 * 2"` bằng 4, vì `"/"` bỏ phần lẻ trước khi nhân.

Line 2 gives 3.0 and 1.0: `"/"` and `"%"` with one float operand give floats. Line 5 shows that `"5 / 2 * 2"` is 5.0 while `"5 // 2 * 2"` is 4, because `"/"` drops the fraction before the multiplication.

Áp dụng vào Sổ Xu: số dư cuối tháng 9 bằng số dư đầu tháng cộng tổng thu trừ tổng chi. Tổng thu gồm một khoản 4.000.000 đồng và ba khoản gia sư 1.200.000 đồng; tổng chi là cộng năm danh mục.

Applied to Sổ Xu: the closing balance for September equals the opening balance plus total income minus total spending. Income is one 4,000,000 dong transfer and three tutoring payments of 1,200,000 dong; spending is the sum of five categories.

```
SO_DU_DAU_THANG = 2_500_000
tong_thu = 4_000_000 + 3 * 1_200_000
tong_chi = 1_800_000 + 585_000 + 120_000 + 449_000 + 180_000
so_du_cuoi = SO_DU_DAU_THANG + tong_thu - tong_chi
print(tong_thu, tong_chi, so_du_cuoi)
```

Kết quả chạy / Output

```
7600000 3134000 6966000
```

2.5. Số thực dấu phẩy động, round và Decimal — *Floating-point numbers, round and Decimal*

Máy lưu **float** theo hệ nhị phân (chuẩn IEEE 754). Nhiều số thập phân như 0.1 không có biểu diễn nhị phân hữu hạn nên được lưu gần đúng; sai số nhỏ này xuất hiện khi tính và khi so sánh.

*The machine stores a **float** in binary (IEEE 754). Many decimal numbers such as 0.1 have no finite binary form and are stored approximately; the tiny error shows up in arithmetic and in comparisons.*

```
print(0.1 + 0.2)
print(0.1 + 0.2 == 0.3)
print(f"{0.1:.20f}")
import math
print(math.isclose(0.1 + 0.2, 0.3))
print(abs((0.1 + 0.2) - 0.3) < 1e-9)
```

Kết quả chạy / Output

```
0.30000000000000004
False
0.10000000000000000555
True
True
```

LỖI HAY GẶP / COMMON MISTAKE — So sánh hai số thực bằng "==". Kết quả của $0.1 + 0.2$ là 0.30000000000000004 nên phép so sánh với 0.3 cho False. Dùng `math.isclose(a, b)` hoặc so sánh $|a - b|$ với một ngưỡng nhỏ.

Comparing two floats with "==". $0.1 + 0.2$ is 0.30000000000000004, so the comparison with 0.3 gives False. Use `math.isclose(a, b)` or compare $|a - b|$ with a small tolerance.

Hàm **round(x, k)** làm tròn đến k chữ số thập phân. Với giá trị nằm đúng giữa, Python làm tròn về số chẵn gần nhất (làm tròn ngân hàng, banker's rounding): `round(2.5)` là 2, `round(3.5)` là 4. Số thực lưu gần đúng nên `round(2.675, 2)` cho 2.67 chứ không phải 2.68.

*The **round(x, k)** function rounds to k decimal places. At an exact halfway value Python rounds to the nearest even number (banker's rounding): `round(2.5)` is 2 and `round(3.5)` is 4. Because floats are approximate, `round(2.675, 2)` gives 2.67, not 2.68.*

```
print(round(2.5), round(3.5), round(0.5), round(1.5))
print(round(3.14159, 2), round(1234.5678, -2))
print(round(2.675, 2))
print(round(1_000_000 * 1.004 ** 3))
```

Kết quả chạy / Output

```
2 4 0 2
3.14 1200.0
2.67
1012048
```

Ví dụ lãi suất của Sổ Xu: lãi 4,8%/năm ghép theo tháng là 0,4%/tháng. Gửi một lần 1.000.000 đồng, sau 3 tháng số tiền là $1.000.000 * (1 + 0,004) ** 3$; kết quả float có nhiều chữ số lẻ nên cần định dạng hoặc làm tròn trước khi in.

An interest example from *Sổ Xu*: 4.8% a year compounded monthly is 0.4% a month. Depositing 1,000,000 dong once, the amount after 3 months is $1,000,000 * (1 + 0.004) ** 3$; the float result has many decimals, so format or round it before printing.

```
von = 1_000_000
lai_thang = 0.048 / 12
sau_3_thang = von * (1 + lai_thang) ** 3
print(sau_3_thang)
print(f"{sau_3_thang:,.2f}")
print(round(sau_3_thang))
```

Kết quả chạy / Output

```
1012048.064
1,012,048.06
1012048
```

Tiêu chí <i>Criterion</i>	float	Decimal	int
Cách lưu <i>Storage</i>	nhị phân, gần đúng <i>binary, approximate</i>	thập phân, đúng <i>decimal, exact</i>	số nguyên, đúng <i>integer, exact</i>
Dùng cho <i>Use for</i>	đo đạc, khoa học <i>measurement, science</i>	tiền, lãi, thuế <i>money, interest, tax</i>	đồng, số lượng <i>whole dong, counts</i>
Khai báo <i>Creating</i>	0.1	Decimal("0.1")	100
Tốc độ <i>Speed</i>	nhanh <i>fast</i>	chậm hơn <i>slower</i>	nhanh <i>fast</i>

Với tiền, hai cách an toàn là tính trên **int** theo đơn vị đồng (không có phần lẻ) hoặc dùng **Decimal** từ module "decimal". Luôn tạo Decimal từ chuỗi: Decimal(0.1) mang theo sai số của float 0.1, còn Decimal("0.1") là đúng 0,1.

For money, two safe options are to compute in **int** dong (no fractions) or to use **Decimal** from the "decimal" module. Always build a Decimal from a string: Decimal(0.1) carries the error of the float 0.1, while Decimal("0.1") is exactly 0.1.

```
from decimal import Decimal
a = Decimal("0.1")
b = Decimal("0.2")
print(a + b)
print(a + b == Decimal("0.3"))
print(Decimal(0.1))
```

Kết quả chạy / Output

```
0.3
True
0.1000000000000000055511151231257827021181583404541015625
```

```
from decimal import Decimal, ROUND_HALF_UP
so_du_dau = Decimal("2500000")
lai = so_du_dau * Decimal("0.004")
print(lai)
print(lai.quantize(Decimal("1"), rounding=ROUND_HALF_UP))
print(Decimal("2.675").quantize(Decimal("0.01"), rounding=ROUND_HALF_UP))
```

Kết quả chạy / Output

```
10000.000
10000
2.68
```

"quantize" đưa số về số chữ số thập phân của mẫu và nhận kiểu làm tròn; ROUND_HALF_UP là kiểu quen thuộc (0,5 làm tròn lên), khác kiểu làm tròn ngân hàng của round().

"quantize" brings a number to the decimal places of a template and accepts a rounding mode; ROUND_HALF_UP is the familiar rule (0.5 rounds up), unlike the banker's rounding of round().

VÍ DỤ / EXAMPLE — Hoàn tiền thẻ tính trên số tiền nguyên đồng: $3_134_000 * 5 // 1000 = 15670$ đồng (0,5%) không có phần lẻ; viết $3_134_000 * 0.005$ cho 15670.0, kiểu float.

*Card cash-back in whole dong: $3_134_000 * 5 // 1000 = 15670$ dong (0.5%) with no fraction; writing $3_134_000 * 0.005$ gives 15670.0, a float.*

2.6. Ép kiểu, input và print — Type conversion, input and print

Ép kiểu (type conversion) đổi giá trị sang kiểu khác bằng int(), float(), str(), bool(). int() cắt phần thập phân; chuỗi chỉ đổi được khi nội dung viết đúng dạng số.

Type conversion turns a value into another type with int(), float(), str() and bool(). int() drops the fractional part; a string converts only when its content is written as a valid number.

```
print(int("45"), float("3.5"), str(12))
print(int(3.99), int(-3.99), float(7))
print(bool(0), bool(""), bool("a"), bool(None))
print(int(" 45 "), int(True), str(None))
print(int(float("3.5")))
```

Kết quả chạy / Output

```
45 3.5 12
3 -3 7.0
False False True False
45 1 None
3
```

LỖI HAY GẶP / COMMON MISTAKE — Đổi chuỗi viết theo kiểu Việt Nam sang số.

int("1.200.000") báo ValueError vì dấu chấm ngăn nghìn không phải cú pháp số của Python; phải bỏ dấu chấm trước. int("3.5") cũng báo lỗi, cần float("3.5") hoặc int(float("3.5")).

Converting a string written in Vietnamese style. int("1.200.000") raises ValueError because the dot as a thousands separator is not Python number syntax; remove the dots first. int("3.5") also fails; use float("3.5") or int(float("3.5")).

```
so_tien = int("1.200.000")
```

Kết quả chạy / Output

```
Traceback (most recent call last):
  File "chuong_trinh.py", line 1, in <module>
    so_tien = int("1.200.000")
              ^^^^^^^^^^^^^^^^^
ValueError: invalid literal for int() with base 10: '1.200.000'
```

```
chuoi = "1.200.000"
so_tien = int(chuoi.replace(".", ""))
print(so_tien + 300_000)
```

Kết quả chạy / Output

```
1500000
```

input(lời_nhắc) hiển thị lời nhắc, chờ người dùng gõ và Enter, rồi **luôn trả về chuỗi (str)** dù người dùng gõ chữ số. Muốn tính toán phải ép kiểu.

input(prompt) displays the prompt, waits for the user to type and press Enter, and always returns a string (str) even if digits were typed. Calculation requires a conversion.

```
ten = input("Họ tên: ")
tuoi = input("Tuổi: ")
print(ten, tuoi, type(tuoi).__name__)
print(int(tuoi) + 1)
```

Kết quả chạy / Output

```
Họ tên: Nguyễn Hoàng Minh
Tuổi: 18
Nguyễn Hoàng Minh 18 str
19
```

LỖI HAY GẶP / COMMON MISTAKE — Quên ép kiểu kết quả của input(). Chuỗi "45000" nhân với 2 không báo lỗi mà cho chuỗi "4500045000" (lỗi logic).

Forgetting to convert the result of input(). The string "45000" times 2 raises no error but gives the string "4500045000" (a logic error).

```
tien = input("Tiền com (đồng): ")
print(tien * 2)
print(int(tien) * 2)
```

Kết quả chạy / Output

```
Tiền com (đồng): 45000
4500045000
90000
```

Một lệnh input() đọc được nhiều giá trị nếu người dùng gõ cách nhau bằng dấu cách: split() tách thành danh sách chuỗi rồi gán nhiều biến. Hàm map(int, ...) ép từng phần tử sang int trong một lệnh.

A single input() can read several values if the user separates them with spaces: split() produces a list of strings that is assigned to several variables. The function map(int, ...) converts every item to int in one statement.

```
thu, chi = input("Thu và chi: ").split()
print(int(thu) - int(chi))
a, b = map(int, input("Hai số: ").split())
print(a + b)
```

Kết quả chạy / Output

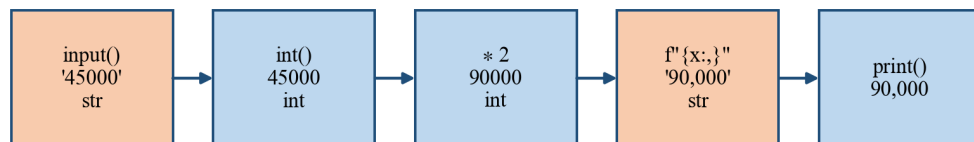
```
Thu và chi: 7600000 3134000
4466000
Hai số: 5 7
12
```

```
print(bool("0"), bool(0.0), bool(" "), bool(""))
print(str(3.0), str(True), float("1e3"))
print(int("-12"), int("+7"), float("-0.5"))
```

Kết quả chạy / Output

```
True False True False
3.0 True 1000.0
-12 7 -0.5
```

Biểu thức <i>Expression</i>	Kết quả <i>Result</i>	Giải thích <i>Explanation</i>
<code>int("42")</code>	42	chuỗi đúng dạng số nguyên <i>a string in integer form</i>
<code>int(3.99)</code>	3	cắt phần thập phân <i>drops the fraction</i>
<code>float(3)</code>	3.0	thêm phần thập phân <i>adds a fractional part</i>
<code>str(3.0)</code>	"3.0"	chuỗi biểu diễn của số <i>the string form of the number</i>
<code>bool("0")</code>	True	chuỗi không rỗng luôn là True <i>any non-empty string is True</i>
<code>bool(0.0)</code>	False	0, 0.0, "", None là False <i>0, 0.0, "" and None are False</i>
<code>int("3.5")</code>	ValueError	chuỗi có dấu chấm không đổi thẳng sang int <i>a string with a dot cannot go straight to int</i>



Hình 2.3. Đường đi của dữ liệu nhập: chuỗi, ép kiểu, tính, định dạng, in
Figure 2.3. Path of input data: string, conversion, calculation, formatting, printing

print(...) nhận nhiều đối số và nối chúng bằng tham số **sep** (mặc định một dấu cách), kết thúc bằng tham số **end** (mặc định xuống dòng).

print(...) takes several arguments and joins them with the *sep* parameter (default: one space), finishing with the *end* parameter (default: a newline).

```
print("a", "b", "c")
print("a", "b", "c", sep="-")
print("Số dư:", end=" ")
print(6_966_000)
print("x", "y", sep="")
print("dòng 1", "dòng 2", sep="\n")
print("Thư mục: C:\SoXu")
print()
print("Hết")
```

Kết quả chạy / Output

```
a b c
a-b-c
Số dư: 6966000
xy
dòng 1
dòng 2
Thư mục: C:\SoXu

Hết
```

Chuỗi thoát <i>Escape</i>	Ý nghĩa <i>Meaning</i>
\n	Xuống dòng <i>New line</i>
\t	Dấu tab <i>Tab</i>
\\	Một dấu gạch chéo ngược <i>One backslash</i>
\" và \'	Dấu nháy trong chuỗi <i>Quote inside a string</i>

PHẢI THUỘC / MEMORISE — `int("45")` đổi chuỗi thành số nguyên; `input()` luôn trả về `str`; `print(a, b, sep="-", end="")` chỉnh dấu ngăn và dấu kết thúc.
`int("45")` turns a string into an integer; `input()` always returns `str`; `print(a, b, sep="-", end="")` sets the separator and the ending.

2.7. Chuỗi định dạng f-string — Formatted string literals (f-strings)

f-string là chuỗi có tiền tố `f`; biểu thức trong ngoặc nhọn được tính và đưa vào chuỗi. Sau dấu hai chấm là **đặc tả định dạng** (format specifier): `"{giá_trị:đặc_tả}"`.

*An **f-string** is a string with the prefix `f`; expressions in curly braces are evaluated and inserted into the string. After the colon comes the **format specifier**: `"{value:specifier}"`.*

```
so_du = 6_966_000
ten = "Minh"
print(f"Chào {ten}, số dư: {so_du}")
print(f"Số dư: {so_du:,} đồng")
print(f"{so_du:>15,}|")
print(f"{ten:<10}|{ten:^10}|{ten:>10}|")
print(f"{0.004:.1%} {3.14159:.2f} {42:05d}")
print(f"{ten=} {so_du + 1}")
```

Kết quả chạy / Output

```
Chào Minh, số dư: 6966000
Số dư: 6,966,000 đồng
        6,966,000|
Minh      |      Minh      |      Minh|
0.4% 3.14 00042
ten='Minh' 6966001
```

Đặc tả <i>Specifier</i>	Ý nghĩa <i>Meaning</i>	Ví dụ <i>Example</i>	Kết quả <i>Result</i>
:,	ngăn nghìn bằng dấu phẩy <i>comma thousands separator</i>	f"{1234567:,}"	1,234,567
:.2f	2 chữ số thập phân <i>2 decimal places</i>	f"{3.14159:.2f}"	3.14
:.1%	phần trăm, 1 chữ số <i>percent, 1 decimal place</i>	f"{0.004:.1%}"	0.4%
:>10	căn phải, rộng 10 <i>right-aligned, width 10</i>	f"{'Xu':>10}"	Xu
:<10	căn trái, rộng 10 <i>left-aligned, width 10</i>	f"{'Xu':<10}"	Xu
:^10	căn giữa, rộng 10 <i>centred, width 10</i>	f"{'Xu':^10}"	Xu
:05d	số nguyên, đệm 0, rộng 5 <i>integer, zero-padded, width 5</i>	f"{42:05d}"	00042

Python định dạng số theo kiểu Anh (1,234,567.50). Muốn kiểu Việt Nam (1.234.567,50) thì đổi dấu bằng "replace". Phải đổi qua một ký tự tạm; nếu thay thẳng dấu phẩy thành dấu chấm trước, hai loại dấu sẽ lẫn nhau.

Python formats numbers the English way (1,234,567.50). To get Vietnamese style (1.234.567,50), swap the marks with "replace". Go through a temporary character; replacing commas by dots first would mix the two kinds of mark.

```
so_du = 6_966_000
s = f"{so_du:,}"
print(s)
print(s.replace(",", "."))
lai = 4_000.5
t = f"{lai:,.2f}"
print(t)
print(t.replace(",", "#").replace(".", ",").replace("#", "."))
ty_le = f"{0.4123684:.1%}"
print(ty_le.replace(".", ","))
```

Kết quả chạy / Output

```
6,966,000
6.966.000
4,000.50
4.000,50
41,2%
```

```
print(f'{\"Danh mục\":<10}{\"Số tiền\":>12}')
print(\"-\" * 22)
print(f'{\"nhà ở\":<10}{1_800_000:>12,}')
print(f'{\"ăn uống\":<10}{585_000:>12,}')
print(f'{\"đi lại\":<10}{120_000:>12,}')
print(f'{\"học tập\":<10}{449_000:>12,}')
print(f'{\"giải trí\":<10}{180_000:>12,}')
```

Kết quả chạy / Output

Danh mục	Số tiền
nhà ở	1,800,000
ăn uống	585,000
đi lại	120,000
học tập	449,000
giải trí	180,000

Bảng trên căn trái cột chữ, căn phải cột số để hàng đơn vị thẳng cột; năm số tiền là chi theo danh mục của tháng 9/2026 trong Sổ Xu.

The table left-aligns the text column and right-aligns the number column so the units line up; the five amounts are the September 2026 spending by category in Sổ Xu.

PHẢI THUỘC / MEMORISE — `f'{x:,}'` ngàn nghìn; `f'{x:.2f}'` hai số lẻ; `f'{x:.1%}'` phần trăm; `f'{s:>10}'` căn phải; sang kiểu Việt Nam: replace.
`f'{x:,}'` thousands; `f'{x:.2f}'` two decimals; `f'{x:.1%}'` percent; `f'{s:>10}'` right-align; Vietnamese style: replace.

VÍ DỤ / EXAMPLE — Phiếu chi in bằng f-string: `f'{ten:<10}{so_tien:>12,}'` giữ cột chữ và cột số thẳng hàng dù độ dài từng dòng khác nhau.

A spending slip printed with an f-string: `f'{ten:<10}{so_tien:>12,}'` keeps the text and number columns aligned however long each line is.

2.8. Chuỗi cơ bản — Basic strings

Chuỗi (str) là dãy ký tự đặt trong cặp nháy đơn hoặc nháy kép. Chuỗi là kiểu **bất biến** (immutable): không sửa được từng ký tự, chỉ tạo chuỗi mới. Phép `+` nối hai chuỗi, phép `*` lặp chuỗi, `len()` cho số ký tự.

*A **string** (str) is a sequence of characters in single or double quotes. Strings are **immutable**: single characters cannot be changed, only new strings created. The `+` operator joins two strings, `*` repeats a string and `len()` gives the number of characters.*

```
a = "Sổ"
b = "Xu"
print(a + " " + b)
print("-" * 20)
print("=" * 5 + " BÁO CÁO " + "=" * 5)
print("Số dư: " + str(6_966_000))
print(len("Sổ Xu"))
```

Kết quả chạy / Output

```
Sổ Xu
-----
===== BÁO CÁO =====
Số dư: 6966000
5
```

LỖI HAY GẶP / COMMON MISTAKE — Nối chuỗi với số bằng "+". "Số dư: " + 6966000 báo TypeError; phải đổi số bằng str() hoặc dùng f-string.

Joining a string and a number with "+". "Số dư: " + 6966000 raises TypeError; convert the number with str() or use an f-string.

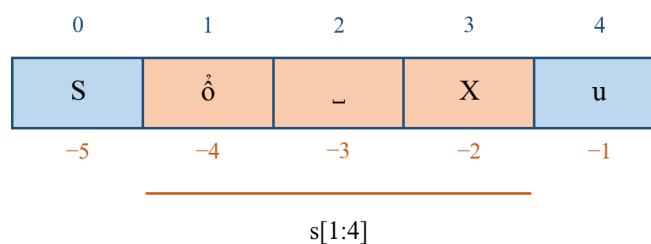
```
so_du = 6_966_000
print("Số dư: " + so_du)
```

Kết quả chạy / Output

```
Traceback (most recent call last):
  File "chuong_trinh.py", line 2, in <module>
    print("Số dư: " + so_du)
    ~~~~~^~~~~~
TypeError: can only concatenate str (not "int") to str
```

Chỉ số (index) đếm từ 0 từ đầu chuỗi; chỉ số âm đếm từ cuối (−1 là ký tự cuối). **Cắt lát (slicing)** s[a:b:c] lấy các ký tự từ chỉ số a đến trước b, bước c; thiếu a thì bắt đầu từ đầu, thiếu b thì đi đến hết. Cắt lát không bao giờ báo IndexError, còn lấy một chỉ số vượt phạm vi thì có.

*The **index** counts from 0 at the start of the string; a negative index counts from the end (−1 is the last character). **Slicing** s[a:b:c] takes the characters from index a up to but not including b, with step c; a missing a starts at the beginning, a missing b runs to the end. A slice never raises IndexError, while a single out-of-range index does.*



Hình 2.4. Chỉ số dương (trên) và chỉ số âm (dưới) của chuỗi "Số Xu"; vùng tô là s[1:4]

Figure 2.4. Positive (above) and negative (below) indices of the string "Số Xu"; the shaded area is s[1:4]

```
ten = "Nguyễn Hoàng Minh"
print(len(ten))
print(ten[0], ten[-1])
print(ten[:6], ten[7:12], ten[-4:])
print(ten[::-1])
s = "Số Xu"
print(s[1:4])
print(s[10:])
```

Kết quả chạy / Output

```
17
N h
Nguyễn Hoàng Minh
hniM gnàoH nễyugN
ỏ X
```

LỖI HAY GẶP / COMMON MISTAKE — Lấy chỉ số vượt phạm vi hoặc sửa từng ký tự của chuỗi. `ten[100]` báo `IndexError`; `s[0] = "X"` báo `TypeError` vì chuỗi bất biến.
Using an out-of-range index or editing a single character. `ten[100]` raises `IndexError`; `s[0] = "X"` raises `TypeError` because strings are immutable.

```
s = "Sở Xu"
print(s[100])
```

Kết quả chạy / Output

```
Traceback (most recent call last):
  File "chuong_trinh.py", line 2, in <module>
    print(s[100])
          ~^^^^^
IndexError: string index out of range
```

```
s = "Sở Xu"
s[0] = "X"
```

Kết quả chạy / Output

```
Traceback (most recent call last):
  File "chuong_trinh.py", line 2, in <module>
    s[0] = "X"
    ~^^^
TypeError: 'str' object does not support item assignment
```

Phương thức <i>Method</i>	Tác dụng <i>Effect</i>	Ví dụ <i>Example</i>
<code>upper()</code> , <code>lower()</code>	đổi hoa, đổi thường <i>to upper / lower case</i>	<code>"chi".upper()</code>
<code>strip()</code>	bỏ khoảng trắng hai đầu <i>trim white space at both ends</i>	<code>" chi ".strip()</code>
<code>title()</code> , <code>capitalize()</code>	viết hoa chữ đầu mỗi từ, chữ đầu câu <i>capitalise each word, the first letter</i>	<code>"nhà ở".title()</code>
<code>split(sep)</code>	tách chuỗi thành danh sách <i>split into a list</i>	<code>"a,b".split(",")</code>
<code>join(ds)</code>	ghép danh sách thành chuỗi <i>join a list into a string</i>	<code>"-".join(["a", "b"])</code>
<code>replace(a, b)</code>	thay mọi a bằng b <i>replace every a with b</i>	<code>"a-b".replace("-", "")</code>
<code>find(x)</code> , <code>count(x)</code>	vị trí đầu tiên (−1 nếu không có), số lần <i>first position (−1 if absent), number of times</i>	<code>"abc".find("b")</code>
<code>startswith(x)</code>	bắt đầu bằng x không <i>starts with x?</i>	<code>"2026".startswith("20")</code>

```
nhap = "  tiền PHÒNG trọ  "
print(nhap.strip())
print(nhap.strip().upper())
print(nhap.strip().lower())
print(nhap.strip().title())
print(nhap.strip().capitalize())
print(len(nhap) - len(nhap.strip()))
```

Kết quả chạy / Output

```
tiền PHÒNG trọ
TIỀN PHÒNG TRỢ
tiền phòng trọ
Tiền Phòng Trọ
Tiền phòng trọ
4
```

Phương thức của chuỗi không sửa chuỗi gốc mà trả về chuỗi mới, nên kết quả phải được gán lại hoặc in ngay. "split" là phương thức đầu tiên của việc đọc dữ liệu có cấu trúc: một dòng văn bản tách ra thành các trường (chương 9 áp dụng với tệp CSV).

String methods do not change the original string; they return a new one, so the result must be assigned or printed at once. "split" is the first step of reading structured data: one line of text becomes separate fields (chapter 9 applies this to CSV files).

```
dong = "2026-09-02,chi,nhà ở,1800000,tiền phòng trọ"
cot = dong.split(",")
print(cot)
print(len(cot), cot[2], int(cot[3]) + 1)
print("2026-09-02".split("-"))
print("/".join(["02", "09", "2026"]))
print(dong.count(","), dong.find("chi"), dong.startswith("2026"))
print("chi" in dong, "thu" in dong)
```

Kết quả chạy / Output

```
['2026-09-02', 'chi', 'nhà ở', '1800000', 'tiền phòng trọ']
5 nhà ở 1800001
['2026', '09', '02']
02/09/2026
4 11 True
True False
```

Chuỗi ba nháy trải trên nhiều dòng và giữ nguyên các dấu xuống dòng. Tiền tố r tạo **chuỗi thô** (raw string): dấu gạch chéo ngược không còn là ký tự thoát, hữu ích với đường dẫn. Máy so sánh chuỗi theo mã Unicode của từng ký tự từ trái sang phải; "ord()" cho mã, "chr()" cho ký tự.

*A triple-quoted string spans several lines and keeps the line breaks. The prefix r makes a **raw string**: the backslash is no longer an escape character, which helps with paths. Strings are compared by the Unicode code of each character from left to right; "ord()" gives the code and "chr()" the character.*

```
phieu = """Phiếu chi
Ngày: 02/09/2026
Số tiền: 1.800.000"""
print(phieu)
print(len(phieu.splitlines()))
duong = r"C:\Users\minh\so_xu"
print(duong)
print(ord("A"), chr(97), ord("a") - ord("A"))
print("minh" < "nam", "Z" < "a", "10" < "9")
```

Kết quả chạy / Output

```
Phiếu chi
Ngày: 02/09/2026
Số tiền: 1.800.000
3
C:\Users\minh\so_xu
65 a 32
True True True
```

"10" < "9" cho True vì ký tự đầu "1" đứng trước "9"; so sánh hai chuỗi không phải so sánh hai số. Ngoài f-string còn hai cách định dạng cũ, vẫn gặp trong mã có sẵn: phương thức format() và toán tử %.

"10" < "9" gives True because the first character "1" comes before "9"; comparing two strings is not comparing two numbers. Besides f-strings there are two older formatting styles still found in existing code: the format() method and the % operator.

```
so_du = 6_966_000
print("Số dư: {:,} đồng".format(so_du))
print("Số dư: %d đồng, lãi %.1f%%" % (so_du, 0.4))
w = 12
print(f"{'Số dư':>{w}}|{'so_du:>{w},'}|")
```

Kết quả chạy / Output

```
Số dư: 6,966,000 đồng
Số dư: 6966000 đồng, lãi 0.4%
Số dư| 6,966,000|
```

PHẢI THUỘC / MEMORISE — s[i] ký tự thứ i (từ 0); s[-1] ký tự cuối; s[a:b] cắt lát; s.split(",") tách; ",".join(ds) ghép; chuỗi bất biến.

s[i] is character i (from 0); s[-1] the last; s[a:b] slices; s.split(",") splits; ",".join(ds) joins; strings are immutable.

LỖI HAY GẶP / COMMON MISTAKE — Dùng tên chưa được gán hoặc gõ sai tên. Python báo NameError ngay khi chạy đến dòng đó; kiểm tra chính tả và thứ tự các dòng.

Using a name that was never assigned, or mistyping it. Python raises NameError as soon as it reaches that line; check the spelling and the order of lines.

```
so_du = 6_966_000
print(so_du)
```

Kết quả chạy / Output

```
Traceback (most recent call last):
  File "chuong_trinh.py", line 2, in <module>
    print(so_du)
          ^^^^^
NameError: name 'so_du' is not defined. Did you mean: 'so_du'?
```

```
thu = 7_600_000
so_thang = 0
print(thu / so_thang)
```

Kết quả chạy / Output

```
Traceback (most recent call last):
  File "chuong_trinh.py", line 3, in <module>
    print(thu / so_thang)
          ~~~~^~~~~~
ZeroDivisionError: division by zero
```

Lỗi <i>Error</i>	Nguyên nhân <i>Cause</i>	Ví dụ <i>Example</i>	Cách sửa <i>Fix</i>
SyntaxError	viết sai cú pháp; không chạy dòng nào <i>wrong syntax; nothing runs</i>	9thang = 1	đổi tên <i>rename</i>
NameError	dùng tên chưa gán <i>name never assigned</i>	print(so_dư)	gán trước, sửa chính tả <i>assign first, fix spelling</i>
TypeError	phép toán sai kiểu <i>operation on the wrong type</i>	"a" + 1	ép kiểu <i>convert the type</i>
ValueError	đúng kiểu nhưng sai giá trị <i>right type, wrong value</i>	int("1.200")	làm sạch chuỗi <i>clean the string</i>
IndexError	chỉ số vượt phạm vi <i>index out of range</i>	s[100]	kiểm tra len(s) <i>check len(s)</i>
ZeroDivisionError	chia cho 0 <i>division by zero</i>	7 / 0	kiểm tra số chia <i>check the divisor</i>
Lỗi logic <i>Logic error</i>	chạy được nhưng sai kết quả <i>runs but gives a wrong result</i>	"45000" * 2	kiểm thử, đọc lại <i>test, re-read</i>

2.9. Sổ Xu bản 0.2: nhập một khoản chi và tính số dư — Sổ Xu version 0.2: entering one expense and computing the balance

Mình mở rộng bản 0.1 (chương 1): chương trình hỏi ngày, danh mục, số tiền và ghi chú của một khoản chi, trừ vào số dư đầu tháng 2.500.000 đồng rồi in phiếu. Chưa có cấu trúc rẽ nhánh, nên bản này chỉ xử lý khoản chi; việc phân biệt thu và chi nằm ở chương 3.

Minh extends version 0.1 (chapter 1): the program asks for the date, category, amount and note of one expense, subtracts it from the opening balance of 2,500,000 dong and prints a slip. Selection structures do not exist yet, so this version handles expenses only; telling income from spending comes in chapter 3.

```
SO_DU_DAU_THANG = 2_500_000
ngay = input("Ngày (aaaa-mm-dd): ")
danh_muc = input("Danh mục: ").strip().lower()
so_tien = int(input("Số tiền chi (đồng): "))
ghi_chu = input("Ghi chú: ").strip()
nam, thang, ng = ngay.split("-")
so_du = SO_DU_DAU_THANG - so_tien
print("=" * 30)
print(f"Ngày      : {ng}/{thang}/{nam}")
print(f"Danh mục   : {danh_muc.capitalize()}")
print(f"Ghi chú    : {ghi_chu.capitalize()}")
tien = f"{so_tien:,}".replace(",", ".")
du = f"{so_du:,}".replace(",", ".")
print(f"Số tiền    : {tien} đ")
print(f"Số dư     : {du} đ")
```

Kết quả chạy / Output

```
Ngày (aaaa-mm-dd): 2026-09-02
Danh mục: NHÀ Ở
Số tiền chi (đồng): 1800000
Ghi chú:   tiền phòng trọ
=====
Ngày      : 02/09/2026
Danh mục  : Nhà ở
Ghi chú   : Tiền phòng trọ
Số tiền   : 1.800.000 đ
Số dư     : 700.000 đ
```

Khoản chi 1.800.000 đồng là tiền phòng trọ ngày 02/9/2026; số dư sau khoản chi là $2.500.000 - 1.800.000 = 700.000$ đồng. Chương trình dùng: input, ép kiểu, split, strip, lower, title, f-string, replace và hằng CHỮ_HOA.

The 1,800,000 dong expense is the boarding-house rent on 02/09/2026; the balance afterwards is 2,500,000 - 1,800,000 = 700,000 dong. The program uses input, conversion, split, strip, lower, title, f-strings, replace and an UPPER_CASE constant.

```
NGAN_SACH = {"học tập": 400_000}
chi_hoc_tap = 350_000 + 99_000
ty_le = chi_hoc_tap / NGAN_SACH["học tập"]
vuot = chi_hoc_tap - NGAN_SACH["học tập"]
print(f"Chi học tập: {chi_hoc_tap:,}")
print(f"So với ngân sách: {ty_le:.2%}")
print(f"Vuốt: {vuot:,}")
```

Kết quả chạy / Output

```
Chi học tập: 449,000
So với ngân sách: 112.25%
Vuốt: 49,000
```

Chi học tập tháng 9 gồm giáo trình 350.000 và khóa học trực tuyến 99.000, tổng 449.000 đồng, bằng 112,25% ngân sách 400.000 đồng và vượt 49.000 đồng. Chương 3 viết phần cảnh báo vượt ngân sách.

September study spending is the textbook at 350,000 plus the online course at 99,000, a total of 449,000 dong, or 112.25% of the 400,000 dong budget, exceeding it by 49,000 dong. Chapter 3 writes the over-budget warning.

VÍ DỤ / EXAMPLE — Chuẩn bị cho chương 3: Số Xu cần so sánh chi_hoc_tap với NGAN_SACH["học tập"]. Hai giá trị 449000 và 400000 đã sẵn sàng; phần còn thiếu là câu trúc if.

Preparing for chapter 3: Số Xu must compare chi_hoc_tap with NGAN_SACH["học tập"]. Both values, 449000 and 400000, are ready; the missing piece is the if structure.

2.10. Bài tập lập trình có lời giải — Worked programming exercises

Bài 1. Đổi số tiền ra các tờ tiền — Exercise 1. Breaking an amount into banknotes

Nhập số tiền (đồng, là bội của 25.000). Tìm số tờ tối thiểu theo thứ tự ưu tiên các mệnh giá 500.000; 200.000; 100.000; 50.000 và số tiền còn lại. Dữ kiện đề bài: số tiền nhập là 3.875.000 đồng.

Read an amount in dong (a multiple of 25,000). Find the number of notes in the order 500,000; 200,000; 100,000; 50,000, and the money left over. Problem data: the amount entered is 3,875,000 dong.

1. Đọc chuỗi nhập và ép sang int.

Read the input and convert it to int.

2. Dùng divmod() lần lượt: thương là số tờ, số dư chuyển sang mệnh giá nhỏ hơn.

Apply divmod() in turn: the quotient is the number of notes, the remainder passes to the next smaller denomination.

3. In bảng có căn lề bằng f-string.

Print an aligned table with f-strings.

```
tien = int(input("Số tiền (đồng): "))
to_500, du = divmod(tien, 500_000)
to_200, du = divmod(du, 200_000)
to_100, du = divmod(du, 100_000)
to_50, du = divmod(du, 50_000)
print(f"{'Mệnh giá':>10}{'Số tờ':>8}")
print(f"{'500.000':>10}{to_500:>8}")
print(f"{'200.000':>10}{to_200:>8}")
print(f"{'100.000':>10}{to_100:>8}")
print(f"{'50.000':>10}{to_50:>8}")
print("Còn lại:", f"{du:,}".replace(",","."))
```

Kết quả chạy / Output

```
Số tiền (đồng): 3875000
Mệnh giá    Số tờ
500.000      7
200.000      1
100.000      1
50.000       1
Còn lại: 25.000
```

3.875.000 gồm 7 tờ 500.000 (3.500.000), còn 375.000; 1 tờ 200.000, còn 175.000; 1 tờ 100.000, còn 75.000; 1 tờ 50.000, còn 25.000. Dùng "/" thay cho "//" cho thương là số thực (7.75), không thể in thành số tờ.

3,875,000 is 7 notes of 500,000 (3,500,000) with 375,000 left; 1 note of 200,000, leaving 175,000; 1 note of 100,000, leaving 75,000; 1 note of 50,000, leaving 25,000. Using "/" instead of "//" gives a float quotient (7.75), which cannot be printed as a number of notes.

Bài 2. Báo cáo thu chi tháng 9 của Sổ Xu — Exercise 2. Sổ Xu spending report for September

Nhập tổng thu và tổng chi tháng 9/2026 của Sổ Xu (7.600.000 và 3.134.000 đồng). Với số dư đầu tháng 2.500.000 đồng, in số dư cuối tháng, số tiền tiết kiệm được trong tháng và tỷ lệ chi trên thu theo kiểu Việt Nam.

Read the September 2026 total income and total spending of Sổ Xu (7,600,000 and 3,134,000 dong). With an opening balance of 2,500,000 dong, print the closing balance, the amount saved during the month and the spending-to-income ratio in Vietnamese style.

1. Khai báo hằng SO_DU_DAU_THANG; đọc hai số và ép sang int.

Declare the constant SO_DU_DAU_THANG; read two numbers and convert them to int.

2. Số dư cuối = đầu + thu – chi; tiết kiệm = thu – chi; tỷ lệ = chi / thu.

Closing balance = opening + income – spending; saved = income – spending; ratio = spending / income.

3. Định dạng bằng f-string rồi đổi dấu bằng replace.

Format with f-strings and swap the marks with replace.

```
SO_DU_DAU_THANG = 2_500_000
thu = int(input("Tổng thu: "))
chi = int(input("Tổng chi: "))
so_du_cuoi = SO_DU_DAU_THANG + thu - chi
tiet_kiem = thu - chi
ty_le = chi / thu
def vn(s):
    return s.replace(",", "#").replace(".", "").replace("#", ".")
print("Số dư cuối tháng :", vn(f"{so_du_cuoi:}"))
print("Tiết kiệm          :", vn(f"{tiet_kiem:}"))
print("Chi trên thu       :", vn(f"{ty_le:.1%}"))
```

Kết quả chạy / Output

```
Tổng thu: 7600000
Tổng chi: 3134000
Số dư cuối tháng : 6.966.000
Tiết kiệm          : 4.466.000
Chi trên thu       : 41,2%
```

Kết quả khớp số liệu Sổ Xu: số dư cuối 6.966.000 đồng; thu trừ chi là 4.466.000 đồng; chi bằng 41,2% thu (3.134.000 / 7.600.000 = 0,4124). Hàm "vn" ở đây dùng trước để gọn mã; cách viết hàm học ở chương 5.

The result matches the Sổ Xu data: closing balance 6,966,000 dong; income minus spending 4,466,000 dong; spending is 41.2% of income (3,134,000 / 7,600,000 = 0.4124). The function "vn" is used early to keep the code short; writing functions is covered in chapter 5.

Bài 3. Điểm tổng kết và làm tròn — Exercise 3. Final mark and rounding

Điểm tổng kết = 10% chuyên cần + 30% giữa kỳ + 60% cuối kỳ, làm tròn đến một chữ số thập phân theo quy tắc 0,05 làm tròn lên. Dữ kiện đề bài: chuyên cần 8,5; giữa kỳ 7,0; cuối kỳ 6,5. So sánh round() trên float với Decimal.

Final mark = 10% attendance + 30% midterm + 60% final exam, rounded to one decimal place with 0.05 rounding up. Problem data: attendance 8.5; midterm 7.0; final exam 6.5. Compare round() on floats with Decimal.

1. Tính bằng float và làm tròn bằng round().
Compute with floats and round with round().
2. Tính lại bằng Decimal tạo từ chuỗi, làm tròn bằng quantize với ROUND_HALF_UP.
Recompute with Decimals built from strings, rounded with quantize and ROUND_HALF_UP.
3. So sánh hai kết quả.
Compare the two results.

```
from decimal import Decimal, ROUND_HALF_UP
cc, gk, ck = 8.5, 7.0, 6.5
dtk = 0.1 * cc + 0.3 * gk + 0.6 * ck
print(dtk, round(dtk, 1))
D = Decimal
dec = D("0.1") * D("8.5") + D("0.3") * D("7.0") + D("0.6") * D("6.5")
print(dec, dec.quantize(D("0.1"), rounding=ROUND_HALF_UP))
```

Kết quả chạy / Output

```
6.85 6.8
6.85 6.9
```

Giá trị đúng là 6,85. Với float, 6.85 lưu hơi thấp hơn 6,85 nên round() cho 6.8; với Decimal, ROUND_HALF_UP cho 6.9, đúng quy tắc đề bài. Điểm số, tiền, thuế nên tính bằng Decimal.

The exact value is 6.85. As a float, 6.85 is stored slightly below 6.85, so round() gives 6.8; with Decimal, ROUND_HALF_UP gives 6.9, as the problem's rule requires. Marks, money and tax are better computed with Decimal.

Bài 4. Tách và chuẩn hóa một dòng giao dịch — Exercise 4. Splitting and cleaning one transaction line

Người dùng dán một dòng giao dịch của Sổ Xu có khoảng trắng và chữ hoa lộn xộn: "2026-09-18, Chi , ĂN UỐNG ,310000, đi chợ". Tách thành năm trường, chuẩn hóa, rồi in dạng chữ thường, số tiền ngàn nghìn kiểu Việt Nam.

The user pastes a Sổ Xu transaction line with untidy spaces and capitals: "2026-09-18, Chi , ĂN UỐNG ,310000, đi chợ". Split it into five fields, clean them, then print lower-case text with the amount in Vietnamese thousands style.

1. Đọc cả dòng bằng input() rồi tách bằng split(",").
Read the whole line with input() and split it with split(",").
2. Với từng trường dùng strip(); lower() cho loại, danh mục, ghi chú; int() cho số tiền.
Apply strip() to each field; lower() to type, category and note; int() to the amount.
3. In bằng f-string; đổi dấu ngàn nghìn.
Print with an f-string; swap the thousands mark.

```

dong = input("Dòng giao dịch: ")
ngay, loai, dm, tien, ghi_chu = dong.split(",")
ngay = ngay.strip()
loai = loai.strip().lower()
dm = dm.strip().lower()
tien = int(tien)
ghi_chu = ghi_chu.strip().lower()
so_tien = f"{tien:,.0f}".replace(".", ",")
print(f"{ngay} | {loai} | {dm} | {so_tien} | {ghi_chu}")
print(len(dong.split(",")), tien + 1)

```

Kết quả chạy / Output

```

Dòng giao dịch: 2026-09-18, Chi , ĂN UỐNG ,310000, đi chợ
2026-09-18 | chi | ăn uống | 310.000 | đi chợ
5 310001

```

`int()` chấp nhận chuỗi " 310000" có khoảng trắng hai đầu, nhưng chuẩn hóa bằng `strip()` vẫn là thói quen tốt với mọi trường chữ. Nếu dòng nhập có sáu trường, lệnh gán năm biến báo `ValueError: too many values to unpack`; xử lý ngoại lệ học ở chương 9.

`int()` accepts a string such as " 310000" with spaces at the ends, but cleaning with `strip()` remains good practice for every text field. If the input line has six fields, assigning to five names raises `ValueError: too many values to unpack`; handling exceptions is covered in chapter 9.

2.11. Bảng thuật ngữ — Glossary

Thuật ngữ <i>Term (Vietnamese)</i>	Tiếng Anh <i>English</i>	Giải thích ngắn <i>Short definition</i>
Biến	variable	Tên gắn với một đối tượng trong bộ nhớ. <i>A name bound to an object in memory.</i>
Phép gán	assignment	Lệnh gắn tên với giá trị của biểu thức. <i>A statement binding a name to the value of an expression.</i>
Hằng	constant	Giá trị không đổi, đặt tên CHỮ_HOÀ theo quy ước. <i>A value that stays fixed, named in UPPER_CASE by convention.</i>
Từ khóa	keyword	Từ Python giữ riêng cho cú pháp, không được đặt làm tên. <i>A word reserved for syntax that cannot be used as a name.</i>
Định danh	identifier	Tên của biến, hàm hoặc đối tượng. <i>The name of a variable, function or object.</i>
Kiểu dữ liệu	data type	Loại giá trị và các phép toán áp dụng được. <i>The kind of value and the operations that apply to it.</i>
Kiểu động	dynamic typing	Kiểu gắn với giá trị, không gắn với tên. <i>The type belongs to the value, not to the name.</i>

Thuật ngữ <i>Term (Vietnamese)</i>	Tiếng Anh <i>English</i>	Giải thích ngắn <i>Short definition</i>
Số nguyên	integer (int)	Số không có phần lẻ, không bị tràn. <i>A number without fraction that never overflows.</i>
Số thực	floating-point number (float)	Số có phần lẻ, lưu nhị phân gần đúng. <i>A number with a fraction, stored approximately in binary.</i>
Chuỗi	string (str)	Dãy ký tự bất biến. <i>An immutable sequence of characters.</i>
Giá trị logic	boolean (bool)	True hoặc False; lớp con của int. <i>True or False; a subclass of int.</i>
Biểu thức	expression	Tổ hợp giá trị, biến, toán tử cho ra một giá trị. <i>Values, variables and operators combined into one value.</i>
Toán tử	operator	Ký hiệu chỉ phép tính, như + hoặc //. <i>A symbol for an operation, such as + or //.</i>
Toán hạng	operand	Giá trị mà toán tử tác động lên. <i>A value that an operator acts on.</i>
Độ ưu tiên	precedence	Thứ tự các phép tính trong một biểu thức. <i>The order in which operations are carried out.</i>
Chia nguyên	floor division	Phép // làm tròn thương xuống. <i>The // operation, rounding the quotient down.</i>
Chia lấy dư	modulo	Phép % cho phần dư của phép chia. <i>The % operation, giving the remainder.</i>
Ép kiểu	type conversion	Đổi giá trị sang kiểu khác bằng int(), str()... <i>Turning a value into another type with int(), str()...</i>
Dấu phẩy động	floating point	Cách lưu số thực theo chuẩn IEEE 754. <i>The IEEE 754 way of storing real numbers.</i>
Làm tròn ngân hàng	banker's rounding	Giá trị giữa làm tròn về số chẵn gần nhất. <i>A halfway value rounds to the nearest even number.</i>
Chuỗi định dạng	f-string	Chuỗi có tiền tố f chứa biểu thức trong ngoặc nhọn. <i>A string with prefix f holding expressions in braces.</i>
Chỉ số	index	Vị trí của ký tự trong chuỗi, đếm từ 0. <i>The position of a character in a string, counted from 0.</i>
Cắt lát	slicing	Lấy một đoạn chuỗi bằng s[a:b:c]. <i>Taking part of a string with s[a:b:c].</i>
Bất biến	immutable	Không sửa được tại chỗ. <i>Cannot be changed in place.</i>

2.12. Câu hỏi tự luận và dàn ý trả lời — Essay questions and answer outlines

Câu 1. Trình bày khái niệm biến và phép gán trong Python; nêu quy tắc đặt tên, vai trò của từ khóa và cách biểu diễn hằng.

Question 1. Explain variables and assignment in Python; state the naming rules, the role of keywords and how constants are represented.

- Biến là tên gắn với một đối tượng; vế phải của phép gán tính trước, ví dụ `so_du = so_du + 4_000_000`.
A variable is a name bound to an object; the right side of an assignment is evaluated first, e.g. "so_du = so_du + 4_000_000".
- Gán nhiều và hoán đổi `"a, b = b, a"`; nêu vì sao không cần biến trung gian.
Multiple assignment and the swap "a, b = b, a"; why no temporary variable is needed.
- Quy tắc đặt tên: chữ cái, chữ số, gạch dưới; không bắt đầu bằng chữ số; phân biệt hoa thường; không trùng từ khóa.
Naming rules: letters, digits, underscores; no leading digit; case-sensitive; not a keyword.
- Hằng không có cú pháp riêng, theo quy ước đặt CHỮ_HOÀ (ví dụ `"SO_DU_DAU_THANG"`); phân biệt `"="` và `"=="`.
Constants have no special syntax and are named in UPPER_CASE by convention (e.g. "SO_DU_DAU_THANG"); distinguish "=" from "==".
- Hai lỗi thường gặp: đặt tên trùng hàm dựng sẵn (sum) và tên bắt đầu bằng chữ số, kèm thông báo lỗi tương ứng.
Two common errors: a name that shadows a built-in (sum) and a name starting with a digit, with the matching error messages.

Câu 2. Trình bày các kiểu dữ liệu cơ bản và các toán tử số học của Python; giải thích vì sao `0.1 + 0.2` không bằng `0.3` và khi nào dùng Decimal.

Question 2. Present Python's basic data types and arithmetic operators; explain why `0.1 + 0.2` does not equal `0.3` and when to use Decimal.

- Năm kiểu cơ bản `int`, `float`, `str`, `bool`, `None`; kiểu động; `int` không tràn số; dùng `type()` và `isinstance()`.
The five basic types int, float, str, bool, None; dynamic typing; int never overflows; using type() and isinstance().
- Các toán tử `+`, `-`, `*`, `/`, `//`, `%`, `**`; `"/"` luôn cho `float`; `"/"` làm tròn xuống, so sánh `-17 // 5` với `int(-17 / 5)`.
*The operators + - * / // % **; "/" always gives a float; "/" rounds down, comparing -17 // 5 with int(-17 / 5).*
- Thứ tự ưu tiên: ngoặc, `**`, dấu đơn ngôi, `*///%`, `+-`; tính `-2 ** 2` và `2 ** 3 ** 2`.
*Precedence: parentheses, **, unary sign, *///%, + -; evaluate -2 ** 2 and 2 ** 3 ** 2.*
- Float lưu nhị phân gần đúng nên `0.1 + 0.2` là `0.30000000000000004`; so sánh bằng `math.isclose()`.
A float is stored as an approximate binary value, so 0.1 + 0.2 is 0.30000000000000004; compare with math.isclose().

- Decimal tạo từ chuỗi cho tiền, lãi, thuế; round() theo kiểu làm tròn ngân hàng, ROUND_HALF_UP của quantize.

Decimal built from strings for money, interest and tax; round() uses banker's rounding, quantize uses ROUND_HALF_UP.

Câu 3. Viết chương trình đọc ngày, danh mục, số tiền của một khoản chi, trừ vào số dư và in phiếu; giải thích từng kỹ thuật nhập, ép kiểu và định dạng.

Question 3. Write a program that reads the date, category and amount of one expense, subtracts it from the balance and prints a slip; explain each input, conversion and formatting technique.

- input() luôn trả về str; ép bằng int() cho số tiền; "45000" * 2 là lỗi logic, int("1.200.000") là ValueError.
*input() always returns str; convert the amount with int(); "45000" * 2 is a logic error, int("1.200.000") is a ValueError.*
- Tách ngày bằng split("-"), chuẩn hóa chữ bằng strip(), lower(), title().
Split the date with split("-"), clean text with strip(), lower() and title().
- Số dư mới = SO_DU_DAU_THANG – so_tien; dùng phép gán kết hợp khi cập nhật nhiều khoản.
New balance = SO_DU_DAU_THANG – so_tien; use augmented assignment when updating over several items.
- In bằng f-string với :, .2f, :>10; đổi 1,800,000 sang 1.800.000 bằng replace qua ký tự tạm.
Print with f-strings using :, .2f, :>10; turn 1,800,000 into 1.800.000 with replace through a temporary character.
- Nêu print(sep, end), chuỗi thoát \n, và kết quả khi chạy với dữ liệu 1.800.000 đồng (số dư còn 700.000 đồng).
State print(sep, end), the escape \n, and the result for 1,800,000 dong (balance left: 700,000 dong).

2.13. Câu hỏi trắc nghiệm — Multiple-choice questions

9. Phát biểu nào sau đây đúng về biến trong Python?

Which statement about variables in Python is correct?

- A.** Phải khai báo kiểu của biến trước khi gán giá trị / *The type of a variable must be declared before assigning a value*
- B.** Biến là tên gắn với một đối tượng và có thể gắn lại sang đối tượng khác kiểu / *A variable is a name bound to an object and can be re-bound to an object of another type*
- C.** Tên biến được phép bắt đầu bằng chữ số / *A variable name may start with a digit*
- D.** Biến viết bằng chữ hoa thì Python không cho gán lại / *Python forbids re-assigning a variable written in capitals*

10. Lệnh "print(7 // 2, 7 % 2)" in ra gì?

What does the statement "print(7 // 2, 7 % 2)" print?

- A. 3 1
- B. 3.5 1
- C. 4 1
- D. 3 0

11. Lệnh "print(-7 // 2)" in ra gì?

What does the statement "print(-7 // 2)" print?

- A. -3
- B. 3
- C. -4
- D. -3.5

12. Lệnh "print(type(10 / 2).__name__)" in ra gì?

What does the statement "print(type(10 / 2).__name__)" print?

- A. int
- B. float
- C. str
- D. NoneType

13. Cách nào an toàn nhất để kiểm tra $0.1 + 0.2$ có xấp xỉ 0.3 hay không?

Which is the safest way to test whether $0.1 + 0.2$ is approximately 0.3 ?

- A. Dùng toán tử == / *Use the == operator*
- B. Dùng toán tử is / *Use the is operator*
- C. Dùng toán tử != / *Use the != operator*
- D. Dùng `math.isclose(0.1 + 0.2, 0.3)` / *Use `math.isclose(0.1 + 0.2, 0.3)`*

14. Lệnh "print(f'{1234567:,.}')" in ra gì?

What does the statement "print(f'{1234567:,.}')" print?

- A. 1.234.567
- B. 1234567
- C. 1 234 567
- D. 1,234,567

15. Lệnh "s = 'Số Xu'; print(s[1:4])" in ra gì?

What does the statement "s = 'Số Xu'; print(s[1:4])" print?

- A. ố X
- B. ố Xu
- C. Số X
- D. Xu

16. Lệnh "print(int('12.5'))" cho kết quả nào?

What is the result of the statement "print(int('12.5'))"?

- A.** In ra 12 / *Prints 12*
- B.** In ra 13 / *Prints 13*
- C.** Báo ValueError / *Raises ValueError*
- D.** In ra 12.5 / *Prints 12.5*

Chương 3. Cấu trúc rẽ nhánh — Chapter 3. Selection structures

Mục tiêu chương — Chapter objectives

- Viết và đánh giá biểu thức điều kiện bằng phép so sánh, so sánh liên tiếp và toán tử and, or, not; đọc bảng chân trị.
Write and evaluate conditions using comparison, chained comparison and the operators and, or, not; read a truth table.
- Dùng if, if-else, if-elif-else, if lồng nhau và biểu thức điều kiện; vẽ lưu đồ rẽ nhánh tương ứng.
Use if, if-else, if-elif-else, nested if and the conditional expression; draw the matching flowchart.
- Dùng match-case (Python 3.10 trở lên) với giá trị đơn, nhiều giá trị nối bằng "|" và mẫu "_".
Use match-case (Python 3.10 or later) with single values, several values joined by "|" and the pattern "_".
- Kiểm tra dữ liệu nhập trước khi tính; nhận biết giá trị truthy và falsy.
Validate input before computing; recognise truthy and falsy values.
- Áp dụng vào Sổ Xu: phân loại giao dịch, tính hoàn tiền theo bậc, cảnh báo vượt ngân sách học tập; tránh ba lỗi điển hình với "=" và "==", thứ tự elif, so sánh số thực.
Apply the tools in Sổ Xu: classify transactions, compute tiered cashback, warn when the study budget is exceeded; avoid three typical errors with "=" and "==", elif order and float comparison.

3.1. Giá trị logic và phép so sánh — Boolean values and comparison

Cấu trúc rẽ nhánh cho chương trình chọn một trong nhiều đường chạy tùy theo điều kiện. Điều kiện là biểu thức có giá trị kiểu bool, chỉ nhận hai giá trị: True (đúng) và False (sai), viết hoa chữ cái đầu. Kiểu bool là lớp con của int: khi tính toán, True bằng 1 và False bằng 0. Mọi phép so sánh trả về một giá trị bool.

A selection structure lets a program choose one of several paths depending on a condition. A condition is an expression of type bool, which takes only two values: True and False, with a capital first letter. The type bool is a subclass of int: in arithmetic, True equals 1 and False equals 0. Every comparison returns a bool value.

```
so_du = 6_966_000
print(so_du > 5_000_000, so_du == 6_966_000, so_du != 0)
print(type(so_du >= 0))
print(True + True, int(False))
```

Kết quả chạy / Output

```
True True True
<class 'bool'>
2 0
```

Toán tử <i>Operator</i>	Ý nghĩa <i>Meaning</i>	Ví dụ <i>Example</i>	Kết quả <i>Result</i>
==	Bằng nhau <i>Equal to</i>	7 == 7.0	True
!=	Khác nhau <i>Not equal to</i>	"chi" != "thu"	True
<	Nhỏ hơn <i>Less than</i>	2 < 10	True
>	Lớn hơn <i>Greater than</i>	2 > 10	False
<=	Nhỏ hơn hoặc bằng <i>Less than or equal to</i>	5 <= 5	True
>=	Lớn hơn hoặc bằng <i>Greater than or equal to</i>	4 >= 5	False
in	Thuộc về (chuỗi con, phần tử) <i>Membership (substring, element)</i>	"xe" in "xe buýt"	True
is None	Là đối tượng None <i>Is the None object</i>	x is None	True nếu x là None <i>True if x is None</i>

Hai số so sánh theo giá trị nên số nguyên và số thực bằng nhau vẫn được coi là bằng nhau. Chuỗi và số là hai kiểu khác nhau nên không bao giờ bằng nhau. Chuỗi so sánh theo thứ tự từng ký tự, mỗi ký tự được so bằng mã Unicode (ord): chữ hoa đứng trước chữ thường, và "2" lớn hơn "10" vì ký tự "2" đứng sau ký tự "1".

Numbers are compared by value, so an integer and a float of equal value are equal. A string and a number are different types and are never equal. Strings are compared character by character using the Unicode code of each character (ord): capital letters come before lower-case letters, and "2" is greater than "10" because the character "2" comes after the character "1".

```
print(5 == 5.0, "5" == 5, "5" == "5.0")
print("2" > "10", 2 > 10)
print("An" < "Bình", "bình" < "Bình", "a" < "B")
print(ord("A"), ord("a"), ord("Đ"), "Đức" > "Zoe")
```

Kết quả chạy / Output

```
True False False
True False
True False False
65 97 272 True
```

PHẢI THUỘC / MEMORISE — So sánh chuỗi dùng mã Unicode của từng ký tự, không theo thứ tự bảng chữ cái tiếng Việt. Chuỗi "9" lớn hơn chuỗi "10"; muốn so theo số phải ép kiểu bằng int() hoặc float() trước.

String comparison uses the Unicode code of each character, not Vietnamese alphabetical order. The string "9" is greater than the string "10"; to compare as numbers, convert with `int()` or `float()` first.

So sánh liên tiếp — *Chained comparison*

Python cho viết nhiều phép so sánh nối nhau, đúng như ký hiệu toán học: " $a < x \leq b$ " tương đương " $a < x$ and $x \leq b$ ", nhưng x chỉ được tính một lần. Dạng này đọc gọn khi kiểm tra một giá trị nằm trong khoảng.

Python allows several comparisons in a row, as in mathematical notation: " $a < x \leq b$ " is the same as " $a < x$ and $x \leq b$ ", but x is evaluated only once. This form reads well when checking that a value lies in an interval.

```
chi = 2_400_000
print(1_000_000 <= chi < 3_000_000)
print(0 < chi <= 2_000_000)
ty_le = 0.005
print(0 <= ty_le <= 1)
```

Kết quả chạy / Output

```
True
False
True
```

3.2. Toán tử logic và giá trị *truthy, falsy* — *Logical operators and truthy and falsy values*

Ba toán tử logic ghép các điều kiện đơn thành điều kiện phức: "and" đúng khi cả hai vế đúng; "or" đúng khi ít nhất một vế đúng; "not" đảo giá trị. Bảng chân trị liệt kê kết quả theo mọi tổ hợp giá trị của hai vế p và q .

Three logical operators combine simple conditions into compound ones: "and" is true when both sides are true; "or" is true when at least one side is true; "not" reverses a value. A truth table lists the result for every combination of values of the two operands p and q .

p	q	p and q	p or q	not p
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

```
print(True and True, True and False, False and False)
print(True or True, True or False, False or False)
print(not True, not False)
```

Kết quả chạy / Output

```
True False False
True True False
False True
```

Ngắn mạch — *Short-circuit evaluation*

Python tính về trái trước và dừng ngay khi đã biết kết quả: "and" dừng ở vế trái sai, "or" dừng ở vế trái đúng. Vế phải không được chạy. Đặc điểm này dùng để chặn lỗi, ví dụ kiểm tra mẫu số khác 0 trước khi chia.

Python evaluates the left operand first and stops as soon as the result is known: "and" stops at a false left operand, "or" stops at a true left operand. The right operand is not run. This is used to prevent errors, for example by checking that a divisor is not zero before dividing.

```
so_luong = 0
tong_tien = 500_000
print(so_luong != 0 and tong_tien / so_luong > 100_000)
print(so_luong == 0 or tong_tien / so_luong > 100_000)
```

Kết quả chạy / Output

```
False
True
```

Đôi thứ tự hai vế của phép "and" đầu tiên thì phép chia chạy trước phép kiểm tra và Python báo lỗi.

Swap the two operands of the first "and" and the division runs before the check, so Python raises an error.

```
so_luong = 0
tong_tien = 500_000
print(tong_tien / so_luong > 100_000 and so_luong != 0)
```

Kết quả chạy / Output

```
Traceback (most recent call last):
  File "chuong_trinh.py", line 3, in <module>
    print(tong_tien / so_luong > 100_000 and so_luong != 0)
          ~~~~~^~~~~~
ZeroDivisionError: division by zero
```

Vì dừng sớm, "and" và "or" trả về chính một trong hai vế, không nhất thiết là True hoặc False. "and" trả về vế trái nếu vế trái là falsy, ngược lại trả về vế phải; "or" trả về vế trái nếu vế trái là truthy, ngược lại trả về vế phải. Cách viết "gia_tri or mac_dinh" thường dùng để gán giá trị mặc định.

Because of early stopping, "and" and "or" return one of the operands itself, not necessarily True or False. "and" returns the left operand if it is falsy and the right operand otherwise; "or" returns the left operand if it is truthy and the right operand otherwise. The form "value or default" is often used to assign a default.

```
so_du = 0
print(so_du and 100)
print(so_du or 100)
print(" " or "chưa đặt tên")
print("Minh" and "đã có tên")
```

Kết quả chạy / Output

```
0
100
chưa đặt tên
đã có tên
```

Thứ tự ưu tiên — Operator precedence

Trong một biểu thức, phép số học tính trước, rồi phép so sánh, rồi "not", rồi "and", cuối cùng "or". Dùng ngoặc tròn để thứ tự tính rõ ràng thay vì dựa vào bảng ưu tiên.

In an expression, arithmetic is evaluated first, then comparisons, then "not", then "and", and finally "or". Use parentheses to make the order explicit instead of relying on the precedence table.

Mức Level	Toán tử Operator	Ghi chú Note
1 (cao nhất) 1 (highest)	()	Ngoặc tròn Parentheses
2	** * / // % + -	Số học, theo bảng chương 2 Arithmetic, as in the table in chapter 2
3	== != < > <= >= in is	So sánh, kể cả so sánh liên tiếp Comparison, including chained comparison
4	not	Đảo giá trị Negation
5	and	Cả hai vế đúng Both sides true
6 (thấp nhất) 6 (lowest)	or	Một trong hai vế đúng Either side true

```
a, b, c = True, False, False
print(a or b and c)
print((a or b) and c)
print(not a == b)
print(not (a and b), (not a) or (not b))
```

Kết quả chạy / Output

```
True
False
True
True True
```

Dòng cuối là hai vế của định luật De Morgan: phủ định của "p and q" bằng "(not p) or (not q)", và phủ định của "p or q" bằng "(not p) and (not q)". Định luật giúp viết lại điều kiện phức cho gọn hoặc dễ đọc.

The last line shows the two sides of De Morgan's law: the negation of "p and q" equals "(not p) or (not q)", and the negation of "p or q" equals "(not p) and (not q)". The law helps rewrite a compound condition more shortly or more readably.

Giá trị truthy và falsy — Truthy and falsy values

Câu lệnh rẽ nhánh chấp nhận mọi giá trị làm điều kiện, kể cả giá trị không phải bool. Python đổi giá trị đó sang bool bằng hàm bool(). Giá trị bị coi là sai (falsy) gồm: False,

None, số 0 và 0.0, chuỗi rỗng, và các tập hợp rỗng như danh sách rỗng (danh sách học ở chương 7). Mọi giá trị còn lại là đúng (truthy).

A selection statement accepts any value as a condition, including values that are not bool. Python converts the value to a bool with bool(). The values treated as false (falsy) are: False, None, the numbers 0 and 0.0, the empty string, and empty collections such as the empty list (lists are covered in chapter 7). Every other value is true (truthy).

```
print(bool(0), bool(0.0), bool(""), bool(None), bool([]))
print(bool(" "), bool("0"), bool(-1), bool([0]))
ghi_chu = ""
print("Có ghi chú" if ghi_chu else "Không có ghi chú")
```

Kết quả chạy / Output

```
False False False False False
True True True True
Không có ghi chú
```

Chuỗi "0" và chuỗi một dấu cách là truthy vì chuỗi chỉ falsy khi rỗng. Với chuỗi người dùng gõ vào, dùng strip() để bỏ dấu cách thừa trước khi kiểm tra rỗng.

The string "0" and a string of one space are truthy because a string is falsy only when empty. For text typed by a user, use strip() to remove extra spaces before testing for emptiness.

3.3. Câu lệnh if và if-else — The if and if-else statements

Câu lệnh "if" chạy một khối lệnh khi điều kiện đúng. Dòng "if" kết thúc bằng dấu hai chấm; các dòng thuộc khối lệnh thụt vào cùng một mức, quy ước 4 dấu cách. Dòng đầu tiên thụt lại về mức cũ là dòng đầu tiên nằm ngoài khối.

The "if" statement runs a block of statements when the condition is true. The "if" line ends with a colon; the lines of the block are indented to the same level, by convention 4 spaces. The first line that returns to the old indentation is the first line outside the block.

```
so_du = 1_250_000
if so_du < 2_000_000:
    print("Số dư thấp, cần chi tiêu thận trọng")
    print("Số dư hiện tại:", so_du)
print("Kết thúc kiểm tra")
```

Kết quả chạy / Output

```
Số dư thấp, cần chi tiêu thận trọng
Số dư hiện tại: 1250000
Kết thúc kiểm tra
```

Dòng "print" cuối không thụt lè nên luôn chạy, dù điều kiện đúng hay sai. Thiếu thụt lè ngay sau dòng "if" gây IndentationError; thụt lè không đều giữa các dòng của cùng một khối cũng gây IndentationError, như ví dụ dưới.

The last "print" line is not indented, so it always runs whether the condition is true or false. Missing indentation right after an "if" line causes an IndentationError, and uneven indentation between lines of the same block also causes an IndentationError, as in the example below.

```
so_du = 1_250_000
if so_du < 2_000_000:
    print("Số dư thấp")
print("Cần chi tiêu thận trọng")
```

Kết quả chạy / Output

```
File "chuong_trinh.py", line 4
print("Cần chỉ tiêu thận trọng")
      ^
IndentationError: unindent does not match any outer indentation level
```

Thêm nhánh "else" để chạy một khối khác khi điều kiện sai. Hai khối loại trừ nhau: mỗi lần chạy, đúng một khối được thực hiện. Ví dụ dưới so sánh khoản chi học tập của Minh với ngân sách 400.000 đồng.

Add an "else" branch to run a different block when the condition is false. The two blocks are mutually exclusive: on each run, exactly one block is executed. The example below compares Minh's study spending with the 400,000 dong budget.

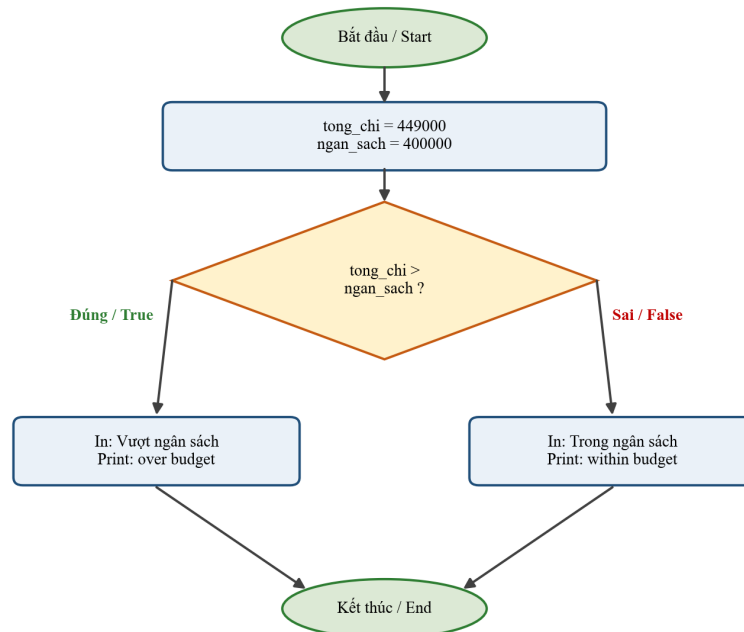
```
tong_chi_hoc_tap = 449_000
ngan_sach = 400_000
if tong_chi_hoc_tap > ngan_sach:
    print("Vượt ngân sách:", tong_chi_hoc_tap - ngan_sach)
else:
    print("Trong ngân sách, còn lại:", ngan_sach - tong_chi_hoc_tap)
```

Kết quả chạy / Output

```
Vượt ngân sách: 49000
```

Khoản chi học tập 449.000 đồng vượt ngân sách 49.000 đồng nên khối "if" chạy và khối "else" bị bỏ qua. Lưu đồ dưới vẽ cùng quá trình: hình thoi là điều kiện, hai mũi tên ra mang nhãn đúng và sai.

The study spending of 449,000 dong exceeds the budget by 49,000 dong, so the "if" block runs and the "else" block is skipped. The flowchart below draws the same process: the diamond is the condition, and the two outgoing arrows are labelled true and false.



Hình 3.1. Lưu đồ rẽ nhánh if-else: so khoản chi học tập với ngân sách

Figure 3.1. Flowchart of an if-else branch: comparing study spending with the budget

Ký hiệu <i>Symbol</i>	Hình <i>Shape</i>	Ý nghĩa <i>Meaning</i>
Bắt đầu, kết thúc <i>Start, end</i>	Hình bầu dục <i>Oval</i>	Điểm vào và điểm ra của thuật toán <i>Entry and exit point of the algorithm</i>
Xử lý, nhập, xuất <i>Process, input, output</i>	Hình chữ nhật <i>Rectangle</i>	Một bước tính hoặc một lệnh <i>One calculation step or one statement</i>
Điều kiện <i>Condition</i>	Hình thoi <i>Diamond</i>	Hai lối ra: đúng và sai <i>Two exits: true and false</i>
Hướng chạy <i>Flow direction</i>	Mũi tên <i>Arrow</i>	Nối các khối theo thứ tự thực hiện <i>Connects the shapes in execution order</i>

if lồng nhau — *Nested if*

Khối lệnh của một "if" có thể chứa "if" khác. Mỗi tầng thụt thêm 4 dấu cách. Dùng khi điều kiện thứ hai chỉ có ý nghĩa sau khi điều kiện thứ nhất đã đúng; nếu các điều kiện độc lập thì ghép bằng "and" cho phẳng hơn.

The block of an "if" can contain another "if". Each level is indented 4 more spaces. Use this when the second condition only makes sense after the first is true; if the conditions are independent, join them with "and" for a flatter layout.

```
so_du = 800_000
loai = "chi"
so_tien = 950_000
if_loai == "chi":
    if so_tien > so_du:
        print("Không đủ số dư, thiếu", so_tien - so_du)
    else:
        so_du -= so_tien
        print("Đã ghi khoản chi, số dư mới:", so_du)
else:
    so_du += so_tien
    print("Đã ghi khoản thu, số dư mới:", so_du)
```

Kết quả chạy / *Output*

```
Không đủ số dư, thiếu 150000
```

```
nam = 2026
thang = 9
if nam > 0:
    if 1 <= thang <= 12:
        print("Tháng hợp lệ")
if nam > 0 and 1 <= thang <= 12:
    print("Viết gọn bằng and")
```

Kết quả chạy / *Output*

```
Tháng hợp lệ
Viết gọn bằng and
```

Khối rỗng và pass — *Empty blocks and pass*

Một khối lệnh không được để trống. Khi chưa biết viết gì, dùng lệnh "pass": lệnh này không làm gì nhưng làm khối hợp lệ về cú pháp, thường dùng để giữ chỗ lúc phác thảo chương trình.

A block must not be empty. When you do not yet know what to write, use the statement "pass": it does nothing but makes the block valid, and is often used as a placeholder while sketching a program.

```
so_du = 3_000_000
if so_du < 0:
    pass
else:
    print("Số dư không âm")
```

Kết quả chạy / Output

```
Số dư không âm
```

3.4. if–elif–else và biểu thức điều kiện — *if–elif–else and the conditional expression*

Khi có nhiều hơn hai khả năng, nối các điều kiện bằng "elif" (else if). Python kiểm tra lần lượt từ trên xuống, chạy khối của điều kiện đúng đầu tiên rồi bỏ qua các nhánh còn lại; "else" ở cuối nhận mọi trường hợp chưa khớp và có thể vắng mặt. Số nhánh "elif" không hạn chế.

When there are more than two possibilities, chain the conditions with "elif" (else if). Python checks them from top to bottom, runs the block of the first true condition and skips the remaining branches; the final "else" takes every case not matched yet and may be omitted. The number of "elif" branches is not limited.

```
diem = 7.5 # dữ kiện đề bài: điểm trung bình học phần
if diem >= 9.0:
    xep_loai = "Xuất sắc"
elif diem >= 8.0:
    xep_loai = "Giỏi"
elif diem >= 7.0:
    xep_loai = "Khá"
elif diem >= 5.0:
    xep_loai = "Trung bình"
else:
    xep_loai = "Yếu"
print(f"Điểm {diem}: {xep_loai}")
```

Kết quả chạy / Output

```
Điểm 7.5: Khá
```

Với điểm 7,5, hai điều kiện đầu sai, điều kiện thứ ba "diem >= 7.0" đúng nên chương trình in Khá và không xét tiếp "diem >= 5.0" dù điều kiện đó cũng đúng. Vì vậy thứ tự các nhánh quyết định kết quả: điều kiện hẹp hơn đặt trước.

With a score of 7.5, the first two conditions are false and the third, "diem >= 7.0", is true, so the program prints Khá (Good) and does not test "diem >= 5.0" even though that is also true. The order of branches therefore decides the result: the narrower condition goes first.

```
diem = 9.2
if diem >= 5.0:
    xep_loai = "Trung bình"
elif diem >= 7.0:
    xep_loai = "Khá"
elif diem >= 9.0:
    xep_loai = "Xuất sắc"
print(f"Điểm {diem}: {xep_loai}")
```

Kết quả chạy / Output

Điểm 9.2: Trung bình

Chương trình trên chạy không báo lỗi nhưng sai: nhánh "diem >= 5.0" đặt đầu nên nuốt mọi điểm từ 5 trở lên. Đây là lỗi logic, Python không phát hiện được. Phép thử đúng là thử các giá trị sát ranh giới (4,9; 5,0; 6,9; 7,0; 8,9; 9,0).

The program above runs without an error message but is wrong: the branch "diem >= 5.0" comes first and swallows every score from 5 upward. This is a logic error that Python cannot detect. The right test is to try values at the boundaries (4.9, 5.0, 6.9, 7.0, 8.9, 9.0).

if liên tiếp và if-elif — Separate if statements and if-elif

Hai câu "if" viết liên nhau là hai câu hỏi độc lập, cả hai đều được kiểm tra. Chuỗi "if-elif" là một câu hỏi nhiều đáp án, chỉ một nhánh chạy.

Two "if" statements written one after the other are two independent questions, and both are tested. An "if-elif" chain is one question with several answers, and only one branch runs.

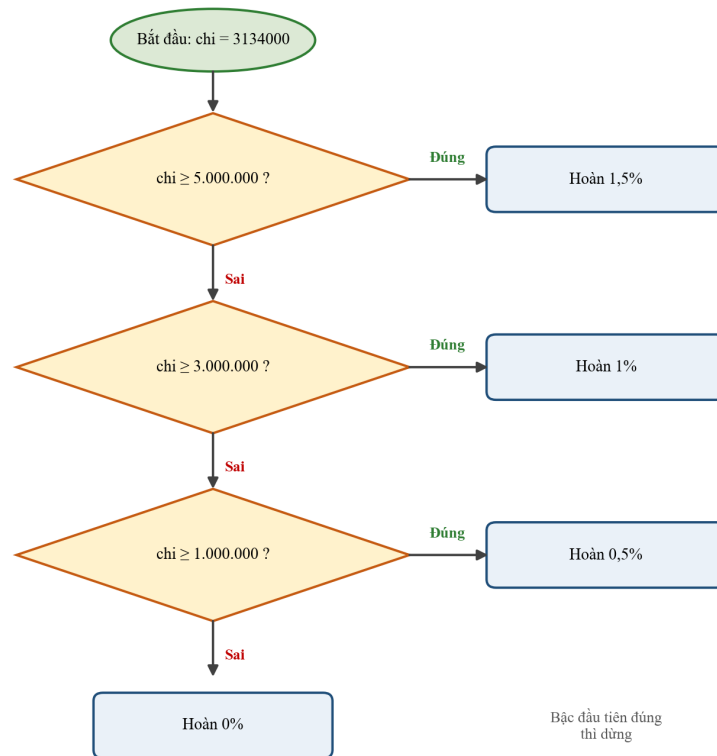
```
diem = 8.5
if diem >= 5:
    print("Đạt")
if diem >= 8:
    print("Giỏi")
print("---")
if diem >= 8:
    print("Giỏi")
elif diem >= 5:
    print("Đạt")
```

Kết quả chạy / Output

Đạt
Giỏi

Giỏi

Tiêu chí Criterion	if liên tiếp Separate if statements	if-elif-else
Số câu hỏi Number of questions	Mỗi "if" là một câu hỏi riêng Each "if" is its own question	Một câu hỏi có nhiều đáp án One question with several answers
Số khối chạy Blocks executed	Từ 0 đến mọi khối From none to all blocks	Nhiều nhất một khối At most one block
Điều kiện sau khi một điều kiện đúng Conditions after one is true	Vẫn được kiểm tra Still tested	Bị bỏ qua Skipped
Dùng khi Use when	Các điều kiện độc lập The conditions are independent	Các trường hợp loại trừ nhau The cases exclude each other



Hình 3.2. Lưu đồ if–elif–else của bảng hoàn tiền theo bậc
Figure 3.2. Flowchart of the if–elif–else chain for tiered cashback

Biểu thức điều kiện — *The conditional expression*

Khi nhánh chỉ để chọn một giá trị, Python có dạng một dòng: "giá_trị_1 if điều_kiện else giá_trị_2". Đây là biểu thức (có giá trị, gán hoặc in được), không phải câu lệnh. Có thể lồng nhưng chỉ nên dùng một hoặc hai tầng.

When a branch only chooses a value, Python has a one-line form: "value_1 if condition else value_2". It is an expression (it has a value that can be assigned or printed), not a statement. It can be nested, but use at most one or two levels.

```

so_tien = 1_800_000
nhân = "chi lớn" if so_tien >= 1_000_000 else "chi nhỏ"
print(nhân)
chenh = -49_000
print("Âm" if chenh < 0 else "Không âm", abs(chenh))
diem = 6.0
print("Giỏi" if diem >= 8 else "Khá" if diem >= 7 else "Còn lại")
    
```

Kết quả chạy / *Output*

```

chi lớn
Âm 49000
Còn lại
    
```

PHẢI THUỘC / MEMORISE — if điều_kiện: ... / elif điều_kiện: ... / else: ... — kiểm tra từ trên xuống, chạy đúng một nhánh đầu tiên có điều kiện đúng. Biểu thức điều kiện: giá_trị_1 if điều_kiện else giá_trị_2. Thứ tự nhánh: hẹp trước, rộng sau.
if condition: ... / elif condition: ... / else: ... — tested from top to bottom, exactly the first branch with

a true condition runs. Conditional expression: value_1 if condition else value_2. Branch order: narrow first, wide last.

3.5. Câu lệnh match–case — The match–case statement

Từ Python 3.10, câu lệnh "match" so một giá trị với lần lượt các mẫu "case" và chạy khối của mẫu khớp đầu tiên. Với so sánh một biến với nhiều giá trị cố định, "match" ngắn và dễ đọc hơn chuỗi "elif" lặp lại phép "==". Mẫu "_" (gạch dưới) khớp mọi giá trị, làm nhánh mặc định và đặt cuối cùng.

From Python 3.10, the "match" statement compares a value with the "case" patterns in turn and runs the block of the first pattern that matches. For comparing one variable with several fixed values, "match" is shorter and easier to read than a chain of "elif" repeating the "==" test. The pattern "_" (underscore) matches any value, serves as the default branch and goes last.

```
loai = "chi"
match loai:
    case "thu":
        print("Khoản thu: cộng vào số dư")
    case "chi":
        print("Khoản chi: trừ khỏi số dư")
    case _:
        print("Loại không hợp lệ")
```

Kết quả chạy / Output

```
Khoản chi: trừ khỏi số dư
```

Nhiều giá trị cùng một nhánh nối bằng dấu gạch đứng "|" (đọc là "hoặc"). Ví dụ dưới gom năm danh mục chi của Sổ Xu thành hai nhóm.

Several values sharing one branch are joined by the vertical bar "|" (read as "or"). The example below groups the five expense categories of Sổ Xu into two groups.

```
danh_muc = "đi lại"
match danh_muc:
    case "nhà ở" | "ăn uống" | "đi lại":
        nhóm = "thiết yếu"
    case "học tập" | "giải trí":
        nhóm = "phát triển và thư giãn"
    case _:
        nhóm = "chưa phân nhóm"
print(f"{danh_muc}: nhóm {nhóm}")
```

Kết quả chạy / Output

```
đi lại: nhóm thiết yếu
```

```
thang = 2
nam = 2026
match thang:
    case 4 | 6 | 9 | 11:
        so_ngay = 30
    case 2:
        nhuan = nam % 4 == 0 and (nam % 100 != 0 or nam % 400 == 0)
        so_ngay = 29 if nhuan else 28
    case _:
        so_ngay = 31
print(f"Tháng {thang}/{nam} có {so_ngay} ngày")
```

Kết quả chạy / Output

```
Tháng 2/2026 có 28 ngày
```

Một nhánh có thể thêm điều kiện bảo vệ (guard) bằng "if" đặt sau mẫu. Mẫu là một tên biến thì khớp mọi giá trị và gán giá trị đó cho tên; guard lọc tiếp trước khi khối chạy.

A branch can add a guard with "if" after the pattern. A pattern that is a variable name matches any value and binds the value to that name; the guard filters further before the block runs.

```
chenh_lech = -49_000
match chenh_lech:
    case 0:
        print("Đúng ngân sách")
    case so if so < 0:
        print("Còn dư ngân sách", -so)
    case so:
        print("Vượt ngân sách", so)
```

Kết quả chạy / Output

```
Còn dư ngân sách 49000
```

Mẫu "case thu:" với "thu" là tên biến không so với biến "thu" đã có; nó là mẫu gán tên, khớp mọi giá trị. Python báo lỗi cú pháp vì các nhánh phía sau không bao giờ được xét. Muốn so với giá trị, viết thẳng giá trị ("thu" trong ngoặc kép) hoặc một tên có dấu chấm.

The pattern "case thu:" where "thu" is a variable name does not compare with the existing variable "thu"; it is a capture pattern that matches any value. Python reports a syntax error because the later branches can never be reached. To compare with a value, write the value itself ("thu" in quotes) or a dotted name.

```
thu = "thu"
loai = "chi"
match loai:
    case thu:
        print("Khoản thu")
    case "chi":
        print("Khoản chi")
```

Kết quả chạy / Output

```
File "chuong_trinh.py", line 4
    case thu:
      ^^^
SyntaxError: name capture 'thu' makes remaining patterns unreachable
```

Tiêu chí Criterion	if–elif–else	match–case
Phiên bản Python <i>Python version</i>	Mọi phiên bản 3 <i>Any version 3</i>	3.10 trở lên <i>3.10 or later</i>
So khoảng giá trị (>=, <) <i>Range tests (>=, <)</i>	Làm trực tiếp <i>Done directly</i>	Phải dùng guard "if" <i>Needs a guard "if"</i>
So một biến với nhiều giá trị cố định <i>One variable against many fixed values</i>	Lặp lại "==" ở mỗi nhánh <i>Repeats "==" in every branch</i>	Mỗi nhánh một mẫu, gọn <i>One pattern per branch, compact</i>
Nhánh mặc định <i>Default branch</i>	else	case _

Tiêu chí <i>Criterion</i>	if–elif–else	match–case
Điều kiện kết hợp nhiều biến <i>Conditions combining several variables</i>	Dùng and, or <i>Uses and, or</i>	Không phù hợp <i>Not suitable</i>

PHẢI THUỘC / MEMORISE — match biến: / case giá_trị_1 | giá_trị_2: / case _: — mẫu khớp đầu tiên thắng; "_" là nhánh mặc định đặt cuối. Tên biến trần trong "case" là mẫu gán tên, khớp mọi giá trị, không phải phép so sánh.

match variable: / case value_1 | value_2: / case _: — the first matching pattern wins; "_" is the default branch and goes last. A bare variable name in "case" is a capture pattern that matches anything; it is not a comparison.

3.6. Kiểm tra dữ liệu nhập — Validating input

Hàm input() luôn trả về chuỗi, và người dùng có thể gõ rỗng, gõ chữ thay vì số, hoặc gõ số ngoài phạm vi. Chương trình phải kiểm tra trước khi ép kiểu: ép "abc" bằng int() gây ValueError và dừng chương trình.

The function input() always returns a string, and the user may type nothing, type letters instead of digits, or type a number outside the allowed range. A program must check before converting: converting "abc" with int() raises a ValueError and stops the program.

```
so_tien = int(input("Nhập số tiền chi (đồng): "))
print("Đã nhận", so_tien)
```

Kết quả chạy / Output

```
Nhập số tiền chi (đồng): bốn mươi lăm nghìn
Traceback (most recent call last):
  File "chuong_trinh.py", line 1, in <module>
    so_tien = int(input("Nhập số tiền chi (đồng): "))
              ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
ValueError: invalid literal for int() with base 10: 'bốn mươi lăm nghìn'
```

Phương thức isdigit() của chuỗi cho True khi chuỗi không rỗng và chỉ gồm chữ số 0–9. Chuỗi có dấu trừ, dấu chấm hoặc dấu cách như "-5", "3.5", " 7" đều cho False, nên số âm và số thực cần cách kiểm tra khác (học ở chương 9 với try–except).

The string method isdigit() gives True when the string is not empty and consists only of the digits 0–9. A string with a minus sign, a dot or a space, such as "-5", "3.5" or " 7", gives False, so negative numbers and floats need another check (covered in chapter 9 with try–except).

```
print("45000".isdigit(), "-5".isdigit(), "3.5".isdigit())
print("",.isdigit(), " 7".isdigit(), " 7 ".strip().isdigit())
```

Kết quả chạy / Output

```
True False False
False False True
```

Quy trình kiểm tra thường theo ba lớp, từ ngoài vào trong: có nhập gì không; có đúng dạng không; có nằm trong phạm vi không. Mỗi lớp là một nhánh của chuỗi "if–elif–else", nhánh "else" cuối cùng xử lý dữ liệu hợp lệ.

Checking usually works in three layers, from outside to inside: was anything typed; is it in the right form; is it within the allowed range. Each layer is a branch of an "if-elif-else" chain, and the final "else" handles valid data.

```
chuoi = input("Nhập số tiền chi (đồng): ").strip()
if chuoi == "":
    print("Chưa nhập gì")
elif not chuoi.isdigit():
    print("Số tiền phải là số nguyên không âm")
elif not 0 < int(chuoi) <= 50_000_000:
    print("Số tiền phải từ 1 đến 50.000.000")
else:
    so_tien = int(chuoi)
    print(f"Hợp lệ: {so_tien:,} đồng")
```

Kết quả chạy / Output

```
Nhập số tiền chi (đồng): 45000
Hợp lệ: 45,000 đồng
```

Dòng đầu của kết quả là lời nhắc kèm chuỗi người dùng gõ (có cả dấu cách thừa, vì "strip()" chỉ chạy sau khi nhận). Thứ tự nhánh quan trọng: phép "int(chuoi)" ở nhánh thứ ba chỉ chạy khi hai nhánh trước sai, tức chuỗi đã là số nguyên, vì chuỗi "elif" dừng ở nhánh đúng đầu tiên.

The first output line is the prompt followed by the string the user typed (including the extra spaces, because "strip()" runs only after the input is received). The order of branches matters: "int(chuoi)" in the third branch runs only when the two earlier branches were false, that is, when the string is already an integer, because an "elif" chain stops at the first true branch.

Kiểm tra giá trị thuộc một tập cho trước dùng toán tử "in". Kiểm tra định dạng ngày "năm-tháng-ngày" dùng độ dài chuỗi, vị trí dấu gạch và phạm vi tháng.

Checking that a value belongs to a given set uses the operator "in". Checking the date format "year-month-day" uses the string length, the positions of the hyphens and the month range.

```
loai = "chi"
ngay = "2026-09-30"
hop_le_loai = loai in ("thu", "chi")
hop_le_ngay = (len(ngay) == 10 and ngay[4] == "-" and ngay[7] == "-"
               and ngay[:4].isdigit() and ngay[5:7].isdigit()
               and 1 <= int(ngay[5:7]) <= 12)
print(hop_le_loai, hop_le_ngay)
if hop_le_loai and hop_le_ngay:
    print("Nhận giao dịch")
```

Kết quả chạy / Output

```
True True
Nhận giao dịch
```

3.7. Ứng dụng trong Sổ Xu — Application in Sổ Xu

Phiên bản 0.3 của Sổ Xu thêm ba tính năng rẽ nhánh: phân loại giao dịch theo loại và độ lớn, tính hoàn tiền thẻ theo bậc chi tiêu, và cảnh báo khi một danh mục vượt ngân sách. Dữ liệu là 12 giao dịch tháng 9/2026 của Nguyễn Hoàng Minh.

Version 0.3 of Sổ Xu adds three selection features: classifying transactions by type and size, computing tiered cashback on card spending, and warning when a category exceeds its budget. The data are the 12 transactions of September 2026 of Nguyễn Hoàng Minh.

Phân loại giao dịch — *Classifying a transaction*

Quy ước phân loại: khoản thu là "khoản thu"; khoản chi từ 1.000.000 đồng là chi lớn, từ 200.000 đồng đến dưới 1.000.000 đồng là chi vừa, dưới 200.000 đồng là chi nhỏ. Giao dịch ngày 02/9 (tiền phòng trọ) là chi lớn.

Classification rule: income is "khoản thu" (income); an expense of 1,000,000 dong or more is a large expense, from 200,000 dong to under 1,000,000 dong a medium expense, and under 200,000 dong a small expense. The transaction of 2 September (room rent) is a large expense.

```
gd = ("2026-09-02", "chi", "nhà ở", 1_800_000, "tiền phòng trọ")
ngay, loai, danh_muc, so_tien, ghi_chu = gd
if loai == "thu":
    nhom = "khoản thu"
elif so_tien >= 1_000_000:
    nhom = "chi lớn"
elif so_tien >= 200_000:
    nhom = "chi vừa"
else:
    nhom = "chi nhỏ"
print(f"{ngay} {danh_muc}: {so_tien:,} đồng, {nhom}")
```

Kết quả chạy / Output

```
2026-09-02 nhà ở: 1,800,000 đồng, chi lớn
```

Áp cùng chuỗi nhánh cho cả 12 giao dịch cần vòng lặp "for" (chương 4); ở đây chỉ dùng để đếm số giao dịch mỗi nhóm.

Applying the same chain of branches to all 12 transactions needs a "for" loop (chapter 4); here it is used only to count the transactions in each group.

```
GIAO_DICH = [
    ("2026-09-01", "thu", "gia đình", 4_000_000, "tiền gia đình gửi"),
    ("2026-09-02", "chi", "nhà ở", 1_800_000, "tiền phòng trọ"),
    ("2026-09-03", "chi", "ăn uống", 45_000, "com trưa"),
    ("2026-09-05", "chi", "đi lại", 120_000, "nap thẻ xe buýt"),
    ("2026-09-07", "thu", "làm thêm", 1_200_000, "gia su tuần 1"),
    ("2026-09-08", "chi", "học tập", 350_000, "giáo trình"),
    ("2026-09-10", "chi", "ăn uống", 230_000, "đi chợ"),
    ("2026-09-14", "thu", "làm thêm", 1_200_000, "gia su tuần 2"),
    ("2026-09-15", "chi", "giải trí", 180_000, "xem phim"),
    ("2026-09-18", "chi", "ăn uống", 310_000, "đi chợ"),
    ("2026-09-21", "thu", "làm thêm", 1_200_000, "gia su tuần 3"),
    ("2026-09-25", "chi", "học tập", 99_000, "khóa học trực tuyến"),
]
thu = lon = vua = nho = 0
for ngay, loai, danh_muc, so_tien, ghi_chu in GIAO_DICH:
    if loai == "thu":
        thu += 1
    elif so_tien >= 1_000_000:
        lon += 1
    elif so_tien >= 200_000:
        vua += 1
    else:
        nho += 1
print(f"thu {thu}, chi lớn {lon}, chi vừa {vua}, chi nhỏ {nho}")
```

Kết quả chạy / Output

```
thu 4, chi lớn 1, chi vừa 3, chi nhỏ 4
```

Kết quả $4 + 1 + 3 + 4 = 12$ giao dịch, khớp số giao dịch của tháng: bốn khoản thu, một chi lớn (tiền phòng), ba chi vừa (230.000; 350.000; 310.000) và bốn chi nhỏ (45.000; 120.000; 180.000; 99.000).

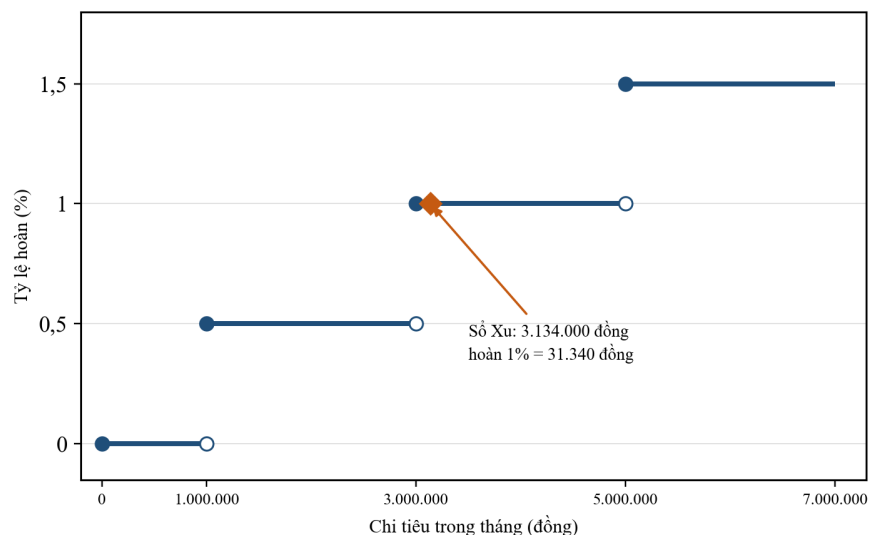
The result $4 + 1 + 3 + 4 = 12$ matches the number of transactions in the month: four incomes, one large expense (the room rent), three medium expenses (230,000; 350,000; 310,000) and four small expenses (45,000; 120,000; 180,000; 99,000).

Hoàn tiền thẻ theo bậc chi tiêu — Tiered card cashback

Thẻ của Minh hoàn tiền theo tổng chi tiêu trong tháng. Tỷ lệ của bậc cao nhất đạt được áp cho toàn bộ số chi, chứ không riêng phần vượt mức. Bốn bậc nằm trong "BAC_HOAN_TIEN", mỗi bậc là cặp (từ mức, tỷ lệ).

Minh's card pays cashback on total monthly spending. The rate of the highest tier reached applies to the whole amount spent, rather than to the part above the threshold. The four tiers are held in "BAC_HOAN_TIEN", each tier a pair (from level, rate).

Bậc Tier	Chi tiêu trong tháng (đồng) Monthly spending (dong)	Tỷ lệ hoàn Cashback rate
1	Dưới 1.000.000 <i>Below 1,000,000</i>	0%
2	Từ 1.000.000 đến dưới 3.000.000 <i>From 1,000,000 to below 3,000,000</i>	0,5% <i>0.5%</i>
3	Từ 3.000.000 đến dưới 5.000.000 <i>From 3,000,000 to below 5,000,000</i>	1%
4	Từ 5.000.000 trở lên <i>5,000,000 or more</i>	1,5% <i>1.5%</i>



Hình 3.3. Tỷ lệ hoàn tiền theo tổng chi tiêu: chấm đặc là giá trị thuộc bậc, chấm rỗng là giá trị bị loại

Figure 3.3. Cashback rate by total spending: a solid dot belongs to the tier, an open dot is excluded

Tổng chi tháng 9 là 3.134.000 đồng, nằm ở bậc 3 (từ 3.000.000 đến dưới 5.000.000), tỷ lệ 1%. Mức hoàn là 31.340 đồng.

Total spending for September is 3,134,000 dong, in tier 3 (from 3,000,000 to below 5,000,000), at a rate of 1%. The cashback is 31,340 dong.

```
tong_chi = 3_134_000
if tong_chi >= 5_000_000:
    ty_le = 0.015
elif tong_chi >= 3_000_000:
    ty_le = 0.01
elif tong_chi >= 1_000_000:
    ty_le = 0.005
else:
    ty_le = 0.0
hoan_tien = round(tong_chi * ty_le)
print(f"Tỷ lệ hoàn: {ty_le:.1%}, hoàn tiền: {hoan_tien:,} đồng")
```

Kết quả chạy / Output

```
Tỷ lệ hoàn: 1.0%, hoàn tiền: 31,340 đồng
```

Thứ tự nhánh đi từ bậc cao xuống bậc thấp, mỗi nhánh chỉ cần một phép ">=" vì nhánh trước đã loại các giá trị lớn hơn. Điểm dễ sai nhất là ranh giới: đúng 3.000.000 đồng thuộc bậc 3 (dùng ">=", không dùng ">"). Bảng sau chạy sáu giá trị sát ranh giới.

The branches run from the highest tier to the lowest, and each needs only one ">=" test because the earlier branch has already excluded larger values. The easiest place to go wrong is the boundary: exactly 3,000,000 dong belongs to tier 3 (use ">=", not ">"). The next run tests six values at the boundaries.

```
for chi in (999_999, 1_000_000, 2_999_999, 3_000_000, 4_999_999, 5_000_000):
    if chi >= 5_000_000:
        ty_le = 0.015
    elif chi >= 3_000_000:
        ty_le = 0.01
    elif chi >= 1_000_000:
        ty_le = 0.005
    else:
        ty_le = 0.0
    print(f"{chi:>9,}   {ty_le:>5.1%}   {round(chi * ty_le):>7,}")
```

Kết quả chạy / Output

```
  999,999   0.0%      0
1,000,000   0.5%    5,000
2,999,999   0.5%   15,000
3,000,000   1.0%   30,000
4,999,999   1.0%   50,000
5,000,000   1.5%   75,000
```

Chi tiêu (đồng) <i>Spending (dong)</i>	Bậc <i>Tier</i>	Tỷ lệ <i>Rate</i>	Tiền hoàn (đồng) <i>Cashback (dong)</i>
999.999 <i>999,999</i>	1	0%	0 <i>0</i>
1.000.000 <i>1,000,000</i>	2	0,5% <i>0.5%</i>	5.000 <i>5,000</i>
2.999.999 <i>2,999,999</i>	2	0,5% <i>0.5%</i>	15.000 <i>15,000</i>

Chi tiêu (đồng) <i>Spending (dong)</i>	Bậc <i>Tier</i>	Tỷ lệ <i>Rate</i>	Tiền hoàn (đồng) <i>Cashback (dong)</i>
3.000.000 <i>3,000,000</i>	3	1%	30.000 <i>30,000</i>
4.999.999 <i>4,999,999</i>	3	1%	50.000 <i>50,000</i>
5.000.000 <i>5,000,000</i>	4	1,5% <i>1.5%</i>	75.000 <i>75,000</i>

Hai giá trị đứng liền nhau qua ranh giới (999.999 và 1.000.000; 2.999.999 và 3.000.000; 4.999.999 và 5.000.000) rơi vào hai bậc khác nhau. Khi kiểm thử, luôn thử cả hai phía của mỗi ranh giới.

Two values on either side of a boundary (999,999 and 1,000,000; 2,999,999 and 3,000,000; 4,999,999 and 5,000,000) fall into different tiers. When testing, always try both sides of every boundary.

Cảnh báo vượt ngân sách học tập — *Warning when the study budget is exceeded*

Ngân sách học tập là 400.000 đồng; tháng 9 Minh chi 350.000 đồng mua giáo trình và 99.000 đồng học trực tuyến, tổng 449.000 đồng. Chương trình tính tỷ lệ đã dùng rồi chia ba mức: trên 100% là vượt, từ 90% là sắp chạm, còn lại là an toàn.

The study budget is 400,000 dong; in September Minh spent 350,000 dong on a textbook and 99,000 dong on an online course, 449,000 dong in all. The program computes the share used and splits it into three levels: above 100% is over budget, from 90% is close to the limit, and the rest is safe.

```

NGAN_SACH = {"nhà ở": 1_800_000, "ăn uống": 1_500_000, "đi lại": 200_000,
             "học tập": 400_000, "giải trí": 300_000}
chi_hoc_tap = 350_000 + 99_000
han_muc = NGAN_SACH["học tập"]
ti_le = chi_hoc_tap / han_muc
if ti_le > 1:
    vuot = chi_hoc_tap - han_muc
    print(f"CẢNH BÁO: học tập vượt {vuot:,} đồng ({ti_le:.2%})")
elif ti_le >= 0.9:
    print(f"Sắp chạm ngân sách học tập ({ti_le:.2%})")
else:
    print(f"Học tập an toàn ({ti_le:.2%})")

```

Kết quả chạy / *Output*

```
CẢNH BÁO: học tập vượt 49,000 đồng (112.25%)
```

Tỷ lệ 112,25% khớp với 449.000 chia 400.000. Biến "NGAN_SACH" là một từ điển (dict) tra ngân sách theo tên danh mục, học kỹ ở chương 8; ở đây chỉ cần hiểu "NGAN_SACH["học tập"]" cho ra 400.000.

The rate of 112.25% matches 449,000 divided by 400,000. The variable "NGAN_SACH" is a dictionary (dict) that looks up the budget by category name, studied in chapter 8; here it is enough to know that "NGAN_SACH["học tập"]" gives 400,000.

3.8. Lỗi hay gặp khi viết rẽ nhánh — *Common mistakes in selection code*

Tiêu chí <i>Criterion</i>	= (phép gán) <i>= (assignment)</i>	== (phép so sánh) <i>== (comparison)</i>
Tác dụng <i>Purpose</i>	Gán giá trị cho biến <i>Stores a value in a variable</i>	Hỏi hai vế có bằng nhau không <i>Asks whether two sides are equal</i>
Kết quả <i>Result</i>	Không có giá trị bool <i>No bool value</i>	True hoặc False <i>True or False</i>
Đặt sau "if" <i>After "if"</i>	SyntaxError <i>SyntaxError</i>	Đúng cú pháp <i>Correct syntax</i>
Ví dụ <i>Example</i>	loai = "chi"	loai == "chi"

LỖI HAY GẶP / COMMON MISTAKE — Viết "if loai = "chi":" thay vì "==". Python báo SyntaxError ngay khi nạp tệp, chưa chạy dòng nào.

Writing "if loai = "chi":" instead of "==". Python raises a SyntaxError as soon as the file is loaded, before any line runs.

```
loai = "chi"
if loai = "chi":
    print("Khoản chi")
```

Kết quả chạy / Output

```
File "chuong_trinh.py", line 2
    if loai = "chi":
    ^^^^^^^^^^^^^
SyntaxError: invalid syntax. Maybe you meant '==' or ':=' instead of '='?
```

LỖI HAY GẶP / COMMON MISTAKE — Quên dấu hai chấm cuối dòng "if", "elif", "else". Python báo SyntaxError: expected ':'.
Forgetting the colon at the end of an "if", "elif" or "else" line. Python raises SyntaxError: expected ':'.

```
so_du = 100_000
if so_du > 0
    print("Số dư dương")
```

Kết quả chạy / Output

```
File "chuong_trinh.py", line 2
    if so_du > 0
    ^
SyntaxError: expected ':'
```

LỖI HAY GẶP / COMMON MISTAKE — Đặt nhánh rộng trước nhánh hẹp trong chuỗi "elif". Chương trình không báo lỗi nhưng cho kết quả sai (đã minh họa ở mục 3.4 với điểm 9,2 bị xếp Trung bình).

Putting a wide branch before a narrow one in an "elif" chain. The program raises no error but gives a wrong result (shown in section 3.4, where a score of 9.2 is rated Trung bình).

LỖI HAY GẶP / COMMON MISTAKE — So sánh số thực bằng "==". Số thực lưu gần đúng nên 0.1 + 0.2 không bằng 0.3; dùng math.isclose() hoặc so sánh chênh lệch với một sai

số nhỏ.

Comparing floats with "==". Floats are stored approximately, so $0.1 + 0.2$ is not equal to 0.3 ; use `math.isclose()` or compare the difference with a small tolerance.

```
import math
tong = 0.1 + 0.2
if tong == 0.3:
    print("Bằng 0,3")
else:
    print("Khác 0,3:", tong)
print(math.isclose(tong, 0.3))
```

Kết quả chạy / Output

```
Khác 0,3: 0.30000000000000004
True
```

LỖI HAY GẶP / COMMON MISTAKE — Viết "`x == 1 or 2`" để kiểm tra `x` bằng 1 hoặc 2. Vế "`2`" là số khác 0 nên luôn `truthy`, biểu thức luôn đúng. Phải viết "`x == 1 or x == 2`" hoặc "`x in (1, 2)`".

Writing "`x == 1 or 2`" to test whether `x` is 1 or 2. The operand "`2`" is non-zero and so always `truthy`, which makes the expression always true. Write "`x == 1 or x == 2`" or "`x in (1, 2)`".

```
danh_muc = "giải trí"
if danh_muc == "ăn uống" or "đi lại":
    print("Chi thiết yếu")
else:
    print("Chi khác")
if danh_muc in ("ăn uống", "đi lại"):
    print("Chi thiết yếu")
else:
    print("Chi khác")
```

Kết quả chạy / Output

```
Chi thiết yếu
Chi khác
```

LỖI HAY GẶP / COMMON MISTAKE — So sánh hai chuỗi số khi chưa ép kiểu. Chuỗi "`9`" lớn hơn chuỗi "`10`" nên chương trình kết luận 9 lớn hơn 10.

Comparing two numeric strings without converting. The string "`9`" is greater than the string "`10`", so the program concludes that 9 is greater than 10.

```
a = input("Số thứ nhất: ")
b = input("Số thứ hai: ")
if a > b:
    print(a, "lớn hơn", b)
if int(a) > int(b):
    print("Theo giá trị số:", a, "lớn hơn", b)
else:
    print("Theo giá trị số:", a, "không lớn hơn", b)
```

Kết quả chạy / Output

```
Số thứ nhất: 9
Số thứ hai: 10
9 lớn hơn 10
Theo giá trị số: 9 không lớn hơn 10
```

Lỗi <i>Mistake</i>	Loại lỗi <i>Kind of error</i>	Python báo không? <i>Does Python report it?</i>	Cách sửa <i>Fix</i>
if loai = "chi":	Cú pháp <i>Syntax</i>	Có, SyntaxError <i>Yes, SyntaxError</i>	Đổi thành "==" <i>Change to "=="</i>
if x > 0 (thiếu ":") <i>if x > 0 (missing ":")</i>	Cú pháp <i>Syntax</i>	Có, SyntaxError <i>Yes, SyntaxError</i>	Thêm dấu hai chấm <i>Add the colon</i>
Thụt lề sai <i>Wrong indentation</i>	Cú pháp <i>Syntax</i>	Có, IndentationError <i>Yes, IndentationError</i>	Thụt 4 dấu cách đều nhau <i>Indent evenly by 4 spaces</i>
Chia khi mẫu số bằng 0 <i>Dividing when the divisor is 0</i>	Lúc chạy <i>Runtime</i>	Có, ZeroDivisionError <i>Yes, ZeroDivisionError</i>	Kiểm tra mẫu số trước, dùng ngắn mạch <i>Check the divisor first, using short-circuit</i>
Thứ tự elif sai <i>Wrong elif order</i>	Logic <i>Logic</i>	Không <i>No</i>	Thử giá trị ở ranh giới <i>Test boundary values</i>
x == 1 or 2	Logic <i>Logic</i>	Không <i>No</i>	Dùng "in" hoặc lặp lại "x ==" <i>Use "in" or repeat "x =="</i>
So sánh float bằng == <i>Comparing floats with ==</i>	Logic <i>Logic</i>	Không <i>No</i>	Dùng math.isclose() <i>Use math.isclose()</i>

3.9. Bảng thuật ngữ — Glossary

Thuật ngữ <i>Term (Vietnamese)</i>	Tiếng Anh <i>English</i>	Giải thích ngắn <i>Short definition</i>
Rẽ nhánh	Selection, branching	Chọn một trong nhiều đường chạy theo điều kiện <i>Choosing one of several paths according to a condition</i>
Điều kiện	Condition	Biểu thức cho giá trị đúng hoặc sai <i>An expression that gives true or false</i>
Kiểu bool	Boolean type	Kiểu dữ liệu chỉ có True và False <i>A data type with only True and False</i>
Toán tử so sánh	Comparison operator	==, !=, <, >, <=, >= cho kết quả bool <i>==, !=, <, >, <=, >= return a bool</i>
So sánh liên tiếp	Chained comparison	Nhiều phép so sánh nối nhau như a < x <= b <i>Several comparisons in a row such as a < x <= b</i>
Toán tử logic	Logical operator	and, or, not ghép hoặc đảo điều kiện <i>and, or, not combine or reverse conditions</i>
Bảng chân trị	Truth table	Bảng kết quả theo mọi tổ hợp giá trị đầu vào <i>A table of results for every combination of inputs</i>

Thuật ngữ <i>Term (Vietnamese)</i>	Tiếng Anh <i>English</i>	Giải thích ngắn <i>Short definition</i>
Ngắn mạch	Short-circuit evaluation	Dừng tính khi đã biết kết quả của and hoặc or <i>Stopping once the result of and or or is known</i>
Định luật De Morgan	De Morgan's laws	Quy tắc đổi phủ định của and thành or và ngược lại <i>Rules turning the negation of and into or and back</i>
Truthy	Truthy	Giá trị được coi là đúng khi dùng làm điều kiện <i>A value treated as true when used as a condition</i>
Falsy	Falsy	Giá trị được coi là sai: False, None, 0, chuỗi rỗng <i>A value treated as false: False, None, 0, empty string</i>
Khối lệnh	Block	Nhóm lệnh thụt lề cùng mức dưới một dòng điều khiển <i>Statements at the same indentation under a control line</i>
Thụt lề	Indentation	Khoảng trắng đầu dòng xác định khối lệnh <i>Leading whitespace that defines a block</i>
Nhánh	Branch	Một khối lệnh của if, elif, else hoặc case <i>One block of an if, elif, else or case</i>
if lồng nhau	Nested if	Câu lệnh if nằm trong khối của if khác <i>An if statement inside the block of another if</i>
Biểu thức điều kiện	Conditional expression	Dạng a if điều_kiện else b cho một giá trị <i>The form a if condition else b that yields a value</i>
Match-case	Match statement	Câu lệnh so giá trị với các mẫu, từ Python 3.10 <i>A statement comparing a value with patterns, from Python 3.10</i>
Mẫu	Pattern	Phần sau case mô tả giá trị cần khớp <i>The part after case describing the value to match</i>
Điều kiện bảo vệ	Guard	Điều kiện if thêm sau mẫu của case <i>An extra if condition after a case pattern</i>
Lưu đồ	Flowchart	Hình biểu diễn thuật toán bằng khối và mũi tên <i>A diagram of an algorithm using shapes and arrows</i>
Giá trị ranh giới	Boundary value	Giá trị nằm đúng hoặc sát điểm đổi nhánh <i>A value at or next to a point where the branch changes</i>

Thuật ngữ <i>Term (Vietnamese)</i>	Tiếng Anh <i>English</i>	Giải thích ngắn <i>Short definition</i>
Kiểm tra dữ liệu nhập	Input validation	Xác nhận dữ liệu đúng dạng và phạm vi trước khi dùng <i>Confirming data has the right form and range before use</i>

3.10. Bài tập lập trình có lời giải — *Programming exercises with solutions*

Bài 1. Xếp loại học lực từ điểm nhập vào — *Exercise 1. Classifying a grade from a typed score*

Viết chương trình nhập điểm trung bình học phần theo thang 10 (dữ kiện đề bài). Chấp nhận dấu phẩy hoặc dấu chấm thập phân. Nếu chuỗi nhập không phải số không âm, in thông báo lỗi; nếu số nằm ngoài khoảng từ 0 đến 10, in thông báo khác; ngược lại xếp loại: từ 9,0 là Xuất sắc, từ 8,0 là Giỏi, từ 7,0 là Khá, từ 5,0 là Trung bình, dưới 5,0 là Yếu. Chạy với dữ liệu nhập "7,8".

Write a program that reads a course average on a 10-point scale (given in the exercise). Accept a comma or a dot as the decimal mark. If the typed text is not a non-negative number, print an error message; if the number lies outside 0 to 10, print a different message; otherwise classify: 9.0 or more is Xuất sắc (Excellent), 8.0 or more is Giỏi (Very Good), 7.0 or more is Khá (Good), 5.0 or more is Trung bình (Average), below 5.0 is Yếu (Weak). Run it with the input "7,8".

1. Đọc chuỗi, bỏ dấu cách thừa bằng `strip()`, đổi dấu phẩy thành dấu chấm bằng `replace()`.
Read the string, remove extra spaces with `strip()`, and turn the comma into a dot with `replace()`.
2. Lớp kiểm tra thứ nhất: bỏ một dấu chấm rồi hỏi `isdigit()`; sai thì báo không phải số.
First check: remove one dot, then ask `isdigit()`; if false, report that it is not a number.
3. Lớp thứ hai: ép sang float và kiểm tra $0 \leq \text{điểm} \leq 10$ bằng so sánh liên tiếp.
Second check: convert to float and test $0 \leq \text{score} \leq 10$ with a chained comparison.
4. Nhánh hợp lệ: chuỗi `if-elif-else` đi từ ngưỡng cao xuống thấp.
Valid branch: an `if-elif-else` chain going from the high threshold to the low one.

```
chuoi = input("Nhập điểm trung bình (0 đến 10): ").strip()
chuoi = chuoi.replace(",", ".")
if not chuoi.replace(".", "", 1).isdigit():
    print("Điểm phải là một số không âm")
elif not 0 <= float(chuoi) <= 10:
    print("Điểm phải nằm trong khoảng 0 đến 10")
else:
    diem = float(chuoi)
    if diem >= 9.0:
        xep_loai = "Xuất sắc"
    elif diem >= 8.0:
        xep_loai = "Giỏi"
    elif diem >= 7.0:
        xep_loai = "Khá"
    elif diem >= 5.0:
        xep_loai = "Trung bình"
    else:
        xep_loai = "Yếu"
    print(f"Điểm {diem}: {xep_loai}")
```

Kết quả chạy / Output

```
Nhập điểm trung bình (0 đến 10): 7,8
Điểm 7.8: Khá
```

Điểm 7,8 đã đổi thành 7.8, lớn hơn 7,0 và nhỏ hơn 8,0 nên xếp Khá. Số in ra theo kiểu Python (dấu chấm), không phải kiểu Việt Nam. Lỗi hay gặp: gọi float() trước khi kiểm tra chuỗi nên chuỗi "abc" làm chương trình dừng bằng ValueError; hoặc dùng ">" thay ">=" nên điểm đúng 8,0 bị xếp Khá.

The score 7,8 became 7.8, which is above 7.0 and below 8.0, so it is rated Khá. The number is printed in Python style (a dot), not Vietnamese style. Common mistakes: calling float() before checking the string, so "abc" stops the program with a ValueError; or using ">" instead of ">=", so a score of exactly 8.0 is rated Khá.

Bài 2. Sổ Xu: hoàn tiền và khoảng cách đến bậc kế tiếp — Exercise 2. Sổ Xu: cashback and the gap to the next tier

Mình nhập tổng chi tiêu trong tháng. Dùng bậc hoàn tiền của Sổ Xu (0 / 0,5% / 1% / 1,5% tại các mốc 0; 1.000.000; 3.000.000; 5.000.000 đồng), in tỷ lệ và số tiền hoàn. Nếu chưa ở bậc cao nhất, in số tiền cần chi thêm để lên bậc kế tiếp; nếu đã ở bậc cao nhất, in "Đã ở bậc cao nhất". Chạy với tổng chi 3.134.000 đồng của tháng 9.

Mình types the total spending of the month. Use the Sổ Xu cashback tiers (0 / 0.5% / 1% / 1.5% at the levels 0; 1,000,000; 3,000,000; 5,000,000 dong) and print the rate and the cashback amount. If not yet in the top tier, print how much more must be spent to reach the next tier; if already in the top tier, print "Đã ở bậc cao nhất" (already in the top tier). Run it with the September total of 3,134,000 dong.

- Kiểm tra chuỗi nhập là số nguyên không âm bằng isdigit().
Check that the typed string is a non-negative integer with isdigit().
- Mỗi nhánh của chuỗi elif gán ba biến cùng lúc: tỷ lệ hiện tại, mốc kế tiếp, tỷ lệ kế tiếp. Bậc cao nhất đặt mốc kế tiếp bằng None.
Each branch of the elif chain assigns three variables at once: the current rate, the next level and the next rate. The top tier sets the next level to None.

3. In tiền hoàn bằng `round(chi * ty_le)`; rồi rẽ nhánh theo "`moc_ke is None`".

Print the cashback with `round(chi * ty_le)`, then branch on "`moc_ke is None`".

```
chuoi = input("Tổng chi tiêu trong tháng (đồng): ").strip()
if not chuoi.isdigit():
    print("Số tiền phải là số nguyên không âm")
else:
    chi = int(chuoi)
    if chi >= 5_000_000:
        ty_le, moc_ke, ty_le_ke = 0.015, None, None
    elif chi >= 3_000_000:
        ty_le, moc_ke, ty_le_ke = 0.01, 5_000_000, 0.015
    elif chi >= 1_000_000:
        ty_le, moc_ke, ty_le_ke = 0.005, 3_000_000, 0.01
    else:
        ty_le, moc_ke, ty_le_ke = 0.0, 1_000_000, 0.005
    tien = round(chi * ty_le)
    print(f"Tỷ lệ hoàn: {ty_le:.1%}, hoàn tiền: {tien:,} đồng")
    if moc_ke is None:
        print("Đã ở bậc cao nhất")
    else:
        con_thieu = moc_ke - chi
        print(f"Chi thêm {con_thieu:,} đồng để lên bậc {ty_le_ke:.1%}")
```

Kết quả chạy / Output

```
Tổng chi tiêu trong tháng (đồng): 3134000
Tỷ lệ hoàn: 1.0%, hoàn tiền: 31,340 đồng
Chi thêm 1,866,000 đồng để lên bậc 1.5%
```

Với 3.134.000 đồng, Minh ở bậc 1%, được hoàn 31.340 đồng và cần chi thêm 1.866.000 đồng để chạm mốc 5.000.000. Lỗi hay gặp: dùng ">" ở mốc nên người chi đúng 5.000.000 đồng chỉ được 1%; quên rằng bậc cao nhất không có mốc kế tiếp nên phép trừ "None - chi" gây `TypeError`.

With 3,134,000 dong, Minh is in the 1% tier, earns 31,340 dong back and needs 1,866,000 dong more to reach 5,000,000. Common mistakes: using ">" at a threshold, so someone who spends exactly 5,000,000 dong gets only 1%; forgetting that the top tier has no next level, so the subtraction "None - chi" raises a `TypeError`.

Bài 3. Số ngày của một tháng — Exercise 3. Number of days in a month

Nhập tháng và năm. Nếu tháng không thuộc 1 đến 12 hoặc năm không dương, báo dữ liệu không hợp lệ. Ngược lại dùng `match-case` in số ngày của tháng đó; tháng 2 có 29 ngày ở năm nhuận (chia hết cho 4 mà không chia hết cho 100, hoặc chia hết cho 400), 28 ngày ở năm thường. Chạy với tháng 2, năm 2028 (dữ kiện đề bài).

Read a month and a year. If the month is not from 1 to 12 or the year is not positive, report invalid data. Otherwise use `match-case` to print the number of days of that month; February has 29 days in a leap year (divisible by 4 but not by 100, or divisible by 400) and 28 days in a common year. Run it with month 2 and year 2028 (given in the exercise).

1. Đọc hai chuỗi, kiểm tra cả hai là số nguyên không âm bằng `isdigit()` rồi ép sang `int`.
Read two strings, check that both are non-negative integers with `isdigit()`, then convert to `int`.
2. Kiểm tra phạm vi: $1 \leq \text{thang} \leq 12$ và $\text{nam} > 0$.
Check the range: $1 \leq \text{thang} \leq 12$ and $\text{nam} > 0$.
3. `match` thang: nhánh 4 | 6 | 9 | 11 cho 30 ngày; nhánh 2 tính năm nhuận; nhánh "_" cho 31 ngày.

match thang: the branch 4 | 6 | 9 | 11 gives 30 days; the branch 2 works out the leap year; the branch "_" gives 31 days.

```
s_thang = input("Tháng: ").strip()
s_nam = input("Năm: ").strip()
if not (s_thang.isdigit() and s_nam.isdigit()):
    print("Tháng và năm phải là số nguyên không âm")
elif not (1 <= int(s_thang) <= 12 and int(s_nam) > 0):
    print("Dữ liệu ngoài phạm vi")
else:
    thang, nam = int(s_thang), int(s_nam)
    match thang:
        case 4 | 6 | 9 | 11:
            so_ngay = 30
        case 2:
            nhuan = nam % 4 == 0 and (nam % 100 != 0 or nam % 400 == 0)
            so_ngay = 29 if nhuan else 28
        case _:
            so_ngay = 31
    print(f"Tháng {thang}/{nam} có {so_ngay} ngày")
```

Kết quả chạy / Output

```
Tháng: 2
Năm: 2028
Tháng 2/2028 có 29 ngày
```

Năm 2028 chia hết cho 4 và không chia hết cho 100 nên là năm nhuận, tháng 2 có 29 ngày. Năm 1900 chia hết cho 4 và cho 100 nhưng không chia hết cho 400 nên không nhuận; năm 2000 chia hết cho 400 nên nhuận. Lỗi hay gặp: chỉ kiểm tra chia hết cho 4 nên xếp nhầm năm 1900; viết "case 4, 6, 9, 11" thay cho dấu gạch đứng, khi đó Python hiểu là mẫu bộ bốn phần tử chứ không phải bốn giá trị rời.

The year 2028 is divisible by 4 and not by 100, so it is a leap year and February has 29 days. The year 1900 is divisible by 4 and by 100 but not by 400, so it is not a leap year; the year 2000 is divisible by 400, so it is. Common mistakes: testing only divisibility by 4, which misclassifies 1900; writing "case 4, 6, 9, 11" instead of the vertical bar, which Python reads as a four-element sequence pattern and not four separate values.

Bài 4. Sổ Xu: ghi khoản chi mới và cảnh báo ngân sách — Exercise 4. Sổ Xu: recording a new expense and warning on the budget

Mình gõ một khoản chi mới gồm loại, danh mục và số tiền. Chương trình kiểm tra: loại phải là "chi"; danh mục phải có trong ngân sách; số tiền phải là số nguyên dương. Hợp lệ thì cộng vào số đã chi của danh mục và cảnh báo: trên 100% ngân sách là vượt, từ 90% là sắp chạm, còn lại là an toàn. Số đã chi trước đó là số liệu tháng 9 của Sổ Xu (học tập đã chi 449.000 đồng). Chạy với dữ liệu nhập: chi, học tập, 51000.

Minh types a new expense with a type, a category and an amount. The program checks that the type is "chi", the category exists in the budget, and the amount is a positive integer. If valid, it adds the amount to what the category has already spent and warns: above 100% of the budget is over, from 90% is close to the limit, and the rest is safe. The amount already spent is the September data of Sổ Xu (study spending 449,000 dong). Run it with the input: chi, học tập, 51000.

1. Chuẩn hóa chuỗi nhập bằng strip() và lower(), rồi kiểm tra ba lớp bằng một chuỗi if-elif.

Normalise the typed strings with strip() and lower(), then run the three checks in one if-elif chain.

2. Ở nhánh "else" (dữ liệu hợp lệ): tính tổng đã chi và tỷ lệ so với ngân sách.

In the "else" branch (valid data): compute the total spent and its ratio to the budget.

3. Rẽ nhánh lồng bên trong theo ba mức cảnh báo.

Branch inside, nested, on the three warning levels.

```
NGAN_SACH = {"nhà ở": 1_800_000, "ăn uống": 1_500_000, "đi lại": 200_000,
             "học tập": 400_000, "giải trí": 300_000}
DA_CHI = {"nhà ở": 1_800_000, "ăn uống": 585_000, "đi lại": 120_000,
          "học tập": 449_000, "giải trí": 180_000}
loai = input("Loại (thu/chi): ").strip().lower()
muc = input("Danh mục: ").strip().lower()
chuoi = input("Số tiền (đồng): ").strip()
if loai != "chi":
    print("Chỉ kiểm ngân sách cho khoản chi")
elif muc not in NGAN_SACH:
    print("Danh mục không có trong ngân sách")
elif not chuoi.isdigit() or int(chuoi) == 0:
    print("Số tiền phải là số nguyên dương")
else:
    tong = DA_CHI[muc] + int(chuoi)
    ti_le = tong / NGAN_SACH[muc]
    if ti_le > 1:
        vuot = tong - NGAN_SACH[muc]
        print(f"Vuốt ngân sách {muc} {vuot:,} đồng ({ti_le:.0%})")
    elif ti_le >= 0.9:
        print(f"Sắp chạm ngân sách {muc} ({ti_le:.0%})")
    else:
        print(f"An toàn: {muc} dùng {ti_le:.0%}")
```

Kết quả chạy / Output

```
Loại (thu/chi): chi
Danh mục: học tập
Số tiền (đồng): 51000
Vuốt ngân sách học tập 100,000 đồng (125%)
```

Số đã chi học tập 449.000 cộng 51.000 là 500.000 đồng, bằng 125% của ngân sách 400.000, vượt 100.000 đồng. Thử thêm: chi ăn uống 100.000 đồng cho tổng 685.000 đồng, bằng 46% ngân sách 1.500.000, ở mức an toàn. Lỗi hay gặp: kiểm tra "muc not in NGAN_SACH" sau khi đã tra "NGAN_SACH[muc]" nên danh mục lạ gây KeyError; thứ tự ba lớp kiểm tra phải đặt trước phép tính.

Study spending already at 449,000 plus 51,000 is 500,000 dong, which is 125% of the 400,000 budget and 100,000 dong over. A further test: 100,000 dong more on food gives 685,000 dong, 46% of the 1,500,000 budget, which is safe. Common mistakes: testing "muc not in NGAN_SACH" after already looking up "NGAN_SACH[muc]", so an unknown category raises a KeyError; the three checks must come before the calculation.

3.11. Câu hỏi tự luận và dàn ý trả lời — Essay questions and answer outlines

Câu 1. Phân biệt phép gán "=" với phép so sánh "==", và trình bày ba toán tử logic and, or, not cùng cơ chế ngắn mạch.

Question 1. Distinguish the assignment "=" from the comparison "==", and explain the three logical operators and, or, not together with short-circuit evaluation.

- **Nêu tác dụng, kết quả của "=" và "=="; cho ví dụ và lỗi khi viết "if x = 1:".**
State the purpose and result of "=" and "=="; give an example and the error caused by "if x = 1:".
- **Lập bảng chân trị của and, or, not cho hai biến p, q.**
Build the truth table of and, or, not for two variables p, q.
- **Nêu thứ tự ưu tiên not, and, or và vì sao nên dùng ngoặc tròn.**
State the precedence of not, and, or and why parentheses are advisable.
- **Giải thích ngắn mạch: vế nào bị bỏ qua khi nào; ví dụ kiểm tra mẫu số khác 0 trước phép chia.**
Explain short-circuiting: which operand is skipped and when; an example checking a divisor before a division.
- **Chỉ ra "and" và "or" trả về một trong hai vế, ví dụ "0 or 100" cho 100.**
Show that "and" and "or" return one of the operands, for example "0 or 100" gives 100.

Câu 2. So sánh if liên tiếp với if–elif–else, nêu vai trò của thứ tự nhánh, và trình bày cách kiểm tra bằng giá trị ranh giới.

Question 2. Compare separate if statements with if–elif–else, explain the role of branch order, and describe testing with boundary values.

- **Nêu sự khác nhau về số câu hỏi và số khối được chạy giữa if liên tiếp và if–elif–else.**
State how separate if statements and if–elif–else differ in the number of questions and of blocks executed.
- **Cho ví dụ xếp loại điểm với hai thứ tự nhánh khác nhau, một đúng một sai, và giải thích vì sao sai không báo lỗi.**
Give a grade classification example with two branch orders, one right and one wrong, and explain why the wrong one raises no error.
- **Viết lại một đoạn if–else gán giá trị thành biểu thức điều kiện một dòng.**
Rewrite an if–else block that assigns a value as a one-line conditional expression.
- **Chọn sáu giá trị ranh giới để kiểm thử bảng hoàn tiền 0 / 0,5% / 1% / 1,5% tại 1.000.000; 3.000.000; 5.000.000 đồng.**
Choose six boundary values to test the cashback table 0 / 0.5% / 1% / 1.5% at 1,000,000; 3,000,000; 5,000,000 dong.
- **Giải thích khi nào dùng if lồng nhau và khi nào ghép bằng and.**
Explain when to use nested if and when to join conditions with and.

Câu 3. Trình bày câu lệnh match–case và việc kiểm tra dữ liệu nhập trước khi tính toán.

Question 3. *Explain the match–case statement and the validation of input before computing.*

- Nêu cú pháp match–case: mẫu giá trị, nhiều giá trị nối bằng "|", mẫu "_" và điều kiện bảo vệ; yêu cầu phiên bản Python.
State the match–case syntax: value patterns, several values joined by "|", the pattern "_" and the guard; the Python version needed.
- Giải thích vì sao "case thu:" với "thu" là tên biến không so sánh với biến đó.
Explain why "case thu:" with "thu" a variable name does not compare against that variable.
- So sánh match–case với if–elif–else: trường hợp nên dùng của mỗi loại.
Compare match–case with if–elif–else: when each is appropriate.
- Trình bày ba lớp kiểm tra dữ liệu nhập (có nhập không, đúng dạng không, trong phạm vi không) và vai trò của isdigit().
Describe the three layers of input checking (anything typed, right form, within range) and the role of isdigit().
- Liệt kê các giá trị falsy và ví dụ dùng chúng làm điều kiện.
List the falsy values and an example of using them as a condition.

3.12. Câu hỏi trắc nghiệm — Multiple-choice questions

17. Lệnh "print(5 == 5.0, '5' == 5)" in ra gì?

What does the statement "print(5 == 5.0, '5' == 5)" print?

- A. True True
- B. True False
- C. False False
- D. False True

18. Lệnh "print(0 or 'Số Xu')" in ra gì?

What does the statement "print(0 or 'Số Xu')" print?

- A. Số Xu
- B. 0
- C. True
- D. False

19. Mẫu nào trong match–case khớp mọi giá trị còn lại?

Which pattern in match–case matches every remaining value?

- A. default:
- B. else:
- C. case _:
- D. case *:

20. Trong chuỗi if–elif–else, khi một điều kiện đã đúng thì điều gì xảy ra với các nhánh phía sau?

In an if–elif–else chain, what happens to the later branches once a condition is true?

- A.** Vẫn được kiểm tra điều kiện / *Their conditions are still tested*
- B.** Chương trình dừng hẳn / *The program stops completely*
- C.** Chỉ nhánh else được chạy / *Only the else branch runs*
- D.** Bị bỏ qua / *They are skipped*

21. Lệnh "x = 15; print('A' if x > 10 else 'B' if x > 5 else 'C')" in ra gì?

What does the statement "x = 15; print('A' if x > 10 else 'B' if x > 5 else 'C')" print?

- A.** A
- B.** B
- C.** C
- D.** Lỗi cú pháp / *A syntax error*

22. Giá trị nào sau đây là falsy?

Which of the following values is falsy?

- A.** Chuỗi "0" / *The string "0"*
- B.** Chuỗi chứa một dấu cách / *A string holding one space*
- C.** Danh sách [0] / *The list [0]*
- D.** Số thực 0.0 / *The float 0.0*

23. Lệnh "n = 10; print(n % 3 == 1 and n // 3 == 3)" in ra gì?

What does the statement "n = 10; print(n % 3 == 1 and n // 3 == 3)" print?

- A.** False
- B.** 1
- C.** True
- D.** Lỗi / *An error*

24. Dòng nào kiểm tra biến loại bằng "chi" đúng cú pháp?

Which line correctly tests whether the variable loai equals "chi"?

- A.** if loai = "chi":
- B.** if loai == "chi":
- C.** if loai === "chi"
- D.** if loai ==== "chi":

Chương 4. Cấu trúc lặp — Chapter 4. Repetition structures

Mục tiêu chương — Chapter objectives

- Viết được vòng lặp "while" với biến đếm, điều kiện lặp và lính canh; vòng lặp "for" với "range" (start, stop, step), kể cả đếm ngược và duyệt chuỗi.
Write a while loop with a counter, a loop condition and a sentinel; write a for loop with range (start, stop, step), including counting down and traversing a string.
- Dùng đúng "break", "continue" và "else" của vòng lặp; xây vòng lặp lồng để in bảng cửu chương và hình tam giác.
Use break, continue and the loop else clause correctly; build nested loops to print a multiplication table and a triangle.
- Áp dụng các mẫu tích lũy, đếm, tìm lớn nhất – nhỏ nhất; nhận biết nguyên nhân và cách tránh vòng lặp vô hạn.
Apply the accumulate, count and find-the-largest or smallest patterns; recognise the causes of an infinite loop and how to avoid it.
- Hoàn thiện Sổ Xu bản 0.4: số tháng đạt mục tiêu tiết kiệm, lịch trả góp laptop 12 kỳ, menu lặp đến khi chọn thoát.
Complete Sổ Xu version 0.4: months needed to reach the savings goal, a 12-instalment laptop schedule, and a menu that repeats until the user quits.
- Đếm số lần lặp và phân biệt độ phức tạp $O(n)$ với $O(n^2)$.
Count loop iterations and tell $O(n)$ from $O(n^2)$.

4.1. Khái niệm vòng lặp — The idea of repetition

Vòng lặp (loop) cho máy thực hiện một khối lệnh nhiều lần mà chỉ viết khối lệnh đó một lần. Mỗi lần chạy hết khối lệnh là một **lần lặp** (iteration). Python có hai câu lệnh lặp: "while" lặp chừng nào một điều kiện còn đúng, dừng khi chưa biết trước số lần; "for" lặp qua từng phần tử của một dãy (sequence), dừng khi số lần đã biết hoặc do dãy quyết định. Khối lệnh được lặp gọi là **thân vòng lặp** và thụt vào 4 dấu cách so với dòng đầu.

*A **loop** makes the computer run a block of statements many times while the block is written only once. One pass through the block is an **iteration**. Python has two looping statements: while repeats as long as a condition stays true and is used when the number of repetitions is not known in advance; for steps through the items of a sequence and is used when the number is known or set by the sequence. The repeated block is the **loop body**, indented by 4 spaces under the header line.*

Ví dụ dưới đây in ba dòng. Cách thứ nhất viết lặp tay ba lệnh "print"; cách thứ hai dùng "for". Với 3 dòng chưa thấy khác biệt, nhưng với 300 dòng thì cách thứ nhất bất khả thi, còn cách thứ hai chỉ đổi một số.

The example below prints three lines. The first version repeats three print calls by hand; the second uses for. With 3 lines the difference is small, but with 300 lines the first version is impractical while the second needs only one number changed.

```
print("Tháng 1")
print("Tháng 2")
print("Tháng 3")
```

Kết quả chạy / Output

```
Tháng 1
Tháng 2
Tháng 3
```

```
for thang in range(1, 4):
    print("Tháng", thang)
```

Kết quả chạy / Output

```
Tháng 1
Tháng 2
Tháng 3
```

Tiêu chí <i>Criterion</i>	while	for
Số lần lặp <i>Number of repetitions</i>	Chưa biết trước, do điều kiện quyết định <i>Not known in advance, set by a condition</i>	Biết trước hoặc bằng số phần tử của dãy <i>Known, or equal to the number of items in the sequence</i>
Điều kiện dừng <i>Stopping rule</i>	Biểu thức bool trở thành False <i>A bool expression becomes False</i>	Dãy hết phần tử <i>The sequence runs out of items</i>
Biến đếm <i>Counter variable</i>	Tự khai báo trước vòng lặp và tự cập nhật trong thân <i>Declared before the loop and updated inside the body</i>	Python tự gán giá trị kế tiếp cho biến lặp <i>Python assigns the next value to the loop variable</i>
Dùng khi <i>Use when</i>	Nhập đến khi hợp lệ, lặp đến khi đạt mục tiêu <i>Re-entering input, repeating until a goal is reached</i>	Duyệt chuỗi, duyệt danh sách, lặp đúng n lần <i>Traversing a string or list, repeating exactly n times</i>

4.2. Vòng lặp while — The while loop

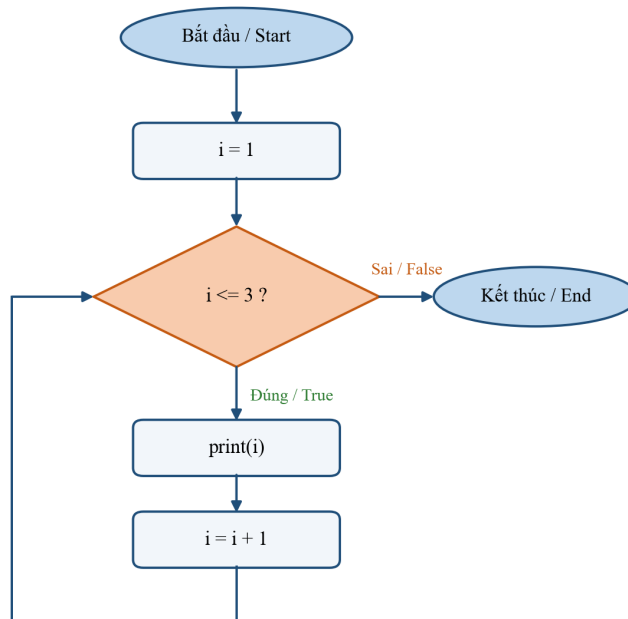
Cú pháp: dòng đầu gồm từ khóa "while", một điều kiện và dấu hai chấm; thân vòng lặp thụt lề bên dưới. Điều kiện được kiểm tra **trước** mỗi lần lặp. Nếu sai ngay lần kiểm tra đầu tiên thì thân vòng lặp không chạy lần nào. Một vòng lặp "while" đúng luôn có bốn thành phần: khởi tạo biến điều khiển, điều kiện, thân, và lệnh cập nhật làm điều kiện tiến dần tới sai.

*Syntax: the header line has the keyword while, a condition and a colon; the loop body is indented below it. The condition is tested **before** every iteration. If it is false at the first test, the body never runs. A correct while loop always has four parts: initialising the control variable, the condition, the body, and an update statement that moves the condition towards false.*

PHẢI THUỘC / MEMORISE — while điều_kiện: thân_vòng_lặp. Điều kiện kiểm tra trước mỗi lần lặp; sai ngay từ đầu thì không lặp lần nào. Bốn thành phần: khởi tạo, điều kiện, thân,

cập nhật.

while condition: loop_body. The condition is tested before each iteration; if false at the start, the body runs zero times. Four parts: initialise, condition, body, update.



Hình 4.1. Lưu đồ vòng lặp "while $i \leq 3$ ": điều kiện kiểm tra trước, thân lặp rồi quay về điều kiện

Figure 4.1. Flowchart of while $i \leq 3$: the condition is tested first, the body runs, then control returns to the condition

Ví dụ đầu tiên đếm từ 1 đến 5. Biến "i" vừa là biến đếm vừa là biến điều khiển: khởi tạo bằng 1, cập nhật bằng "i += 1". Khi "i" bằng 6 thì điều kiện "i <= 5" sai và vòng lặp kết thúc; biến "i" giữ giá trị 6 sau vòng lặp.

The first example counts from 1 to 5. The variable i is both the counter and the control variable: it starts at 1 and is updated by $i += 1$. When i reaches 6 the condition $i \leq 5$ is false and the loop ends; i keeps the value 6 after the loop.

```

i = 1
while i <= 5:
    print(i, end=" ")
    i += 1
print()
print("Sau vòng lặp, i =", i)
    
```

Kết quả chạy / Output

```

1 2 3 4 5
Sau vòng lặp, i = 6
    
```

Điều kiện sai ngay từ đầu thì thân không chạy. Đây là điểm khác với một số ngôn ngữ có vòng lặp "do ... while" (chạy thân ít nhất một lần); Python không có câu lệnh đó.

If the condition is false from the start, the body does not run. This differs from languages that have a do ... while loop (the body runs at least once); Python has no such statement.

```
so_lan = 0
while so_lan > 0:
    print("Dòng này không bao giờ được in")
print("Xong")
```

Kết quả chạy / Output

Xong

Tích lũy là mẫu quen thuộc nhất của "while": một biến khởi tạo 0, mỗi lần lặp cộng thêm một giá trị. Đoạn mã dưới cộng các số từ 1 đến 100; kết quả khớp công thức $100 \times 101 / 2$.

Accumulation is the most familiar use of while: a variable starts at 0 and each iteration adds a value to it. The code below adds the numbers 1 to 100; the result matches the formula $100 \times 101 / 2$.

```
tong = 0
i = 1
while i <= 100:
    tong += i
    i += 1
print("Tổng 1 đến 100 =", tong)
```

Kết quả chạy / Output

Tổng 1 đến 100 = 5050

Khi chưa biết trước số lần lặp, vòng lặp dừng nhờ một **lính canh** (sentinel): giá trị đặc biệt do người dùng nhập để báo hết dữ liệu. Đoạn mã đọc các khoản chi cho đến khi gặp 0. Lệnh "input" đầu tiên đứng trước vòng lặp để điều kiện có giá trị kiểm tra; lệnh "input" thứ hai ở cuối thân để đọc giá trị kế tiếp.

*When the number of repetitions is unknown, the loop stops at a **sentinel**: a special value the user types to say that the data has ended. The code reads expenses until it meets 0. The first input call sits before the loop so that the condition has a value to test; the second input call at the end of the body reads the next value.*

```
tong = 0
so_khoan = 0
tien = int(input("Số tiền chi (0 để dừng): "))
while tien != 0:
    tong += tien
    so_khoan += 1
    tien = int(input("Số tiền chi (0 để dừng): "))
print(f"{so_khoan} khoản, tổng {tong:,} đồng")
```

Kết quả chạy / Output

```
Số tiền chi (0 để dừng): 45000
Số tiền chi (0 để dừng): 120000
Số tiền chi (0 để dừng): 230000
Số tiền chi (0 để dừng): 0
3 khoản, tổng 395,000 đồng
```

Mẫu thứ ba là **nhập lại đến khi hợp lệ**. Điều kiện của "while" mô tả trường hợp dữ liệu còn sai; vì "or" ngắn mạch nên "int(tien)" chỉ được gọi khi "tien.isdigit()" đã đúng. Ba lần nhập dưới đây lần lượt là chữ, số 0 và số hợp lệ.

*The third pattern is **re-entering until valid**. The condition of while describes the case where the data is still wrong; because of short-circuits, int(tien) is called only when tien.isdigit() is already true. The three entries below are a word, the number 0 and a valid number.*

```

tien = input("Số tiền chi: ")
while not tien.isdigit() or int(tien) <= 0:
    print("Không hợp lệ, nhập lại.")
    tien = input("Số tiền chi: ")
print("Đã nhận", int(tien), "đồng")

```

Kết quả chạy / Output

```

Số tiền chi: abc
Không hợp lệ, nhập lại.
Số tiền chi: 0
Không hợp lệ, nhập lại.
Số tiền chi: 45000
Đã nhận 45000 đồng

```

Mẫu while <i>While pattern</i>	Điều kiện lặp <i>Loop condition</i>	Ví dụ <i>Example</i>
Biến đếm <i>Counter</i>	$i \leq n$	Đếm 1 đến n ; tổng các số <i>Counting 1 to n; summing numbers</i>
Lính canh <i>Sentinel</i>	$gia_tri \neq 0$	Nhập đến khi gặp 0 hoặc -1 <i>Reading input until 0 or -1 appears</i>
Nhập lại <i>Re-entry</i>	du_lieu_sai	Hỏi lại đến khi hợp lệ <i>Asking again until the input is valid</i>
Cờ <i>Flag</i>	con_tiep	Biến bool đổi thành False khi đủ điều kiện dừng <i>A bool variable set to False when the stopping rule is met</i>

LỖI HAY GẶP / COMMON MISTAKE — Lệch một lần lặp (off-by-one): viết " $i < 5$ " thay vì " $i \leq 5$ " thì bỏ sót giá trị cuối. Không có thông báo lỗi, chỉ có kết quả sai; cách kiểm là thử với n nhỏ (1, 2) và đối chiếu số lần lặp.

Off-by-one error: writing $i < 5$ instead of $i \leq 5$ drops the last value. There is no error message, only a wrong result; test with a small n (1, 2) and compare the number of iterations.

```

i = 1
while i < 5:
    print(i, end=" ")
    i += 1
print()

```

Kết quả chạy / Output

```
1 2 3 4
```

4.3. Vòng lặp for và hàm range — The for loop and the range function

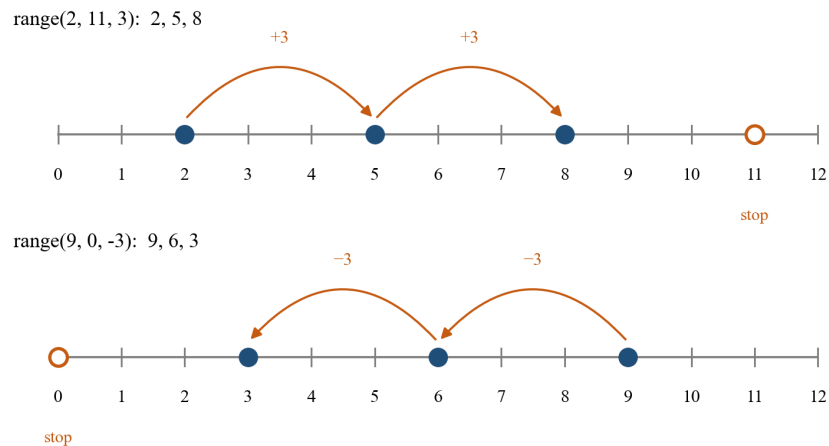
Cú pháp "for biến in dãy:" gán lần lượt từng phần tử của dãy cho biến lặp rồi chạy thân vòng lặp. Python tự cập nhật biến lặp nên không có lệnh tăng biến đếm. Dãy có thể là chuỗi, danh sách, hoặc đối tượng "range". "range" sinh dãy số nguyên theo ba dạng: "range(stop)", "range(start, stop)" và "range(start, stop, step)". Giá trị "stop" **không** được lấy; "start" mặc định 0; "step" mặc định 1 và có thể âm.

The syntax for variable in sequence: assigns each item of the sequence to the loop variable in turn and runs the body. Python updates the loop variable itself, so there is no counter-increment

statement. The sequence can be a string, a list or a range object. range produces integers in three forms: range(stop), range(start, stop) and range(start, stop, step). The value stop is **not** included; start defaults to 0; step defaults to 1 and may be negative.

PHẢI THUỘC / MEMORISE — range(stop) cho 0 đến stop – 1; range(start, stop) cho start đến stop – 1; range(start, stop, step) nhảy cách step. Giá trị stop không bao giờ được lấy. Số phần tử của range(n) là n.

range(stop) gives 0 to stop – 1; range(start, stop) gives start to stop – 1; range(start, stop, step) jumps by step. The value stop is never produced. range(n) has n items.



Hình 4.2. Hai dãy "range": ba số đầu tiên (chấm đặc) và giá trị "stop" không được lấy (vòng tròn rỗng)

Figure 4.2. Two range sequences: the produced values (filled dots) and the stop value, which is not produced (open circle)

```
print(list(range(5)))
print(list(range(2, 6)))
print(list(range(1, 10, 3)))
print(list(range(5, 0, -1)))
print(list(range(5, 5)))
print(len(range(10, 0, -2)))
```

Kết quả chạy / Output

```
[0, 1, 2, 3, 4]
[2, 3, 4, 5]
[1, 4, 7]
[5, 4, 3, 2, 1]
[]
5
```

Biểu thức Expression	Các giá trị Values	Ghi chú Remark
range(5)	0, 1, 2, 3, 4	5 phần tử, không có 5 5 items, 5 is not included
range(2, 6)	2, 3, 4, 5	Từ start đến stop – 1 From start to stop – 1

Biểu thức <i>Expression</i>	Các giá trị <i>Values</i>	Ghi chú <i>Remark</i>
<code>range(1, 10, 3)</code>	1, 4, 7	Giá trị kế tiếp 10 không lấy <i>The next value, 10, is not produced</i>
<code>range(5, 0, -1)</code>	5, 4, 3, 2, 1	Đếm ngược; step âm thì start phải lớn hơn stop <i>Counting down; with a negative step, start must be larger than stop</i>
<code>range(5, 5)</code>	(rỗng) <i>(empty)</i>	Thân for không chạy lần nào <i>The for body never runs</i>

Đếm ngược dùng bước âm. Đoạn mã dưới in từ 5 xuống 1 rồi báo hiệu kết thúc.

Counting down uses a negative step. The code below prints from 5 down to 1 and then signals the end.

```
for i in range(5, 0, -1):
    print(i, end=" ")
print("Xuất phát!")
```

Kết quả chạy / Output

```
5 4 3 2 1 Xuất phát!
```

Tích lũy bằng "for" gọn hơn "while" vì không cần khởi tạo và cập nhật biến đếm. Tổng các số chẵn từ 2 đến 100 dùng bước 2; kết quả 2.550.

Accumulating with for is shorter than with while because the counter needs no initialisation or update. The sum of the even numbers from 2 to 100 uses step 2; the result is 2,550.

```
tong = 0
for so in range(2, 101, 2):
    tong += so
print("Tổng chẵn từ 2 đến 100 =", tong)
```

Kết quả chạy / Output

```
Tổng chẵn từ 2 đến 100 = 2550
```

Chuỗi là một dãy ký tự nên "for" duyệt được từng ký tự. Đoạn mã dưới tách dấu chấm ngăn cách hàng nghìn ra khỏi chuỗi "1.800.000" để đổi thành số nguyên.

A string is a sequence of characters, so for can traverse it one character at a time. The code below separates the thousands dots from the string "1.800.000" so that it can be converted to an integer.

```
chuoi = "1.800.000"
so_cham = 0
chu_so = ""
for ky_tu in chuoi:
    if ky_tu == ".":
        so_cham += 1
    else:
        chu_so += ky_tu
print(so_cham, chu_so, int(chu_so))
```

Kết quả chạy / Output

```
2 1800000 1800000
```

Duyệt danh sách giao dịch của Sổ Xu: mỗi phần tử là một bộ năm giá trị (tuple) và có thể tách thẳng vào năm biến ở dòng "for". Biến "_" nhận giá trị ghi chú không dùng đến. Đoạn mã in các khoản chi từ 200.000 đồng trở lên.

Traversing the Sổ Xu transaction list: each item is a tuple of five values and can be unpacked straight into five variables in the for line. The variable _ receives the note, which is not used. The code prints the expenses of 200,000 dong or more.

```
SO_DU_DAU_THANG = 2_500_000
GIAO_DICH = [
    ("2026-09-01", "thu", "gia đình", 4_000_000, "tiền gia đình gửi"),
    ("2026-09-02", "chi", "nhà ở", 1_800_000, "tiền phòng trọ"),
    ("2026-09-03", "chi", "ăn uống", 45_000, "com trưa"),
    ("2026-09-05", "chi", "đi lại", 120_000, " nạp thẻ xe buýt"),
    ("2026-09-07", "thu", "làm thêm", 1_200_000, "gia su tuần 1"),
    ("2026-09-08", "chi", "học tập", 350_000, "giáo trình"),
    ("2026-09-10", "chi", "ăn uống", 230_000, "đi chợ"),
    ("2026-09-14", "thu", "làm thêm", 1_200_000, "gia su tuần 2"),
    ("2026-09-15", "chi", "giải trí", 180_000, "xem phim"),
    ("2026-09-18", "chi", "ăn uống", 310_000, "đi chợ"),
    ("2026-09-21", "thu", "làm thêm", 1_200_000, "gia su tuần 3"),
    ("2026-09-25", "chi", "học tập", 99_000, "khóa học trực tuyến"),
]
for ngay, loai, danh_muc, tien, _ in GIAO_DICH:
    if loai == "chi" and tien >= 200_000:
        print(ngay, danh_muc, f"{tien:,}")
```

Kết quả chạy / Output

```
2026-09-02 nhà ở 1,800,000
2026-09-08 học tập 350,000
2026-09-10 ăn uống 230,000
2026-09-18 ăn uống 310,000
```

LỖI HAY GẶP / COMMON MISTAKE — Tưởng "range(1, 5)" gồm cả 5. Muốn có 1 đến 5 phải viết "range(1, 6)"; công thức an toàn là "range(1, n + 1)" cho 1 đến n.

Assuming range(1, 5) includes 5. To get 1 to 5 write range(1, 6); the safe form is range(1, n + 1) for 1 to n.

```
print(list(range(1, 5)))
n = 5
print(list(range(1, n + 1)))
```

Kết quả chạy / Output

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

4.4. break, continue và else của vòng lặp — break, continue and the loop else clause

Hai lệnh điều khiển thay đổi nhịp chạy bình thường của vòng lặp. "break" thoát hẳn khỏi vòng lặp chứa nó, chương trình chạy tiếp ở lệnh đứng sau vòng lặp. "continue" bỏ phần còn lại của lần lặp hiện tại và chuyển sang lần lặp kế tiếp. Cả hai chỉ tác động lên vòng lặp trong cùng chứa chúng.

Two control statements change the normal rhythm of a loop. break leaves the loop that contains it, and the program continues at the statement after the loop. continue skips the rest of the current iteration and moves on to the next one. Both affect only the innermost loop that contains them.

Tiêu chí <i>Criterion</i>	break	continue
Tác dụng <i>Effect</i>	Thoát hẳn vòng lặp <i>Leaves the loop entirely</i>	Bỏ qua phần còn lại của lần lặp này <i>Skips the rest of this iteration</i>
Sau lệnh <i>Afterwards</i>	Chạy lệnh đứng sau vòng lặp <i>Runs the statement after the loop</i>	Quay lại kiểm tra điều kiện (while) hoặc lấy phần tử kế (for) <i>Returns to the condition test (while) or takes the next item (for)</i>
Dùng khi <i>Use when</i>	Đã tìm thấy kết quả, không cần xét tiếp <i>The result is found and nothing more needs checking</i>	Phần tử hiện tại không thuộc diện xử lý <i>The current item is not to be processed</i>

Ví dụ "break": tìm khoản chi đầu tiên lớn hơn 1.000.000 đồng. Khi tìm thấy, in kết quả và dừng; các giao dịch phía sau không còn được xét.

A break example: find the first expense above 1,000,000 dong. Once found, print it and stop; the later transactions are no longer examined.

```
SO_DU_DAU_THANG = 2_500_000
GIAO_DICH = [
    ("2026-09-01", "thu", "gia đình", 4_000_000, "tiền gia đình gửi"),
    ("2026-09-02", "chi", "nhà ở", 1_800_000, "tiền phòng trọ"),
    ("2026-09-03", "chi", "ăn uống", 45_000, "cơm trưa"),
    ("2026-09-05", "chi", "đi lại", 120_000, "nạp thẻ xe buýt"),
    ("2026-09-07", "thu", "làm thêm", 1_200_000, "gia su tuần 1"),
    ("2026-09-08", "chi", "học tập", 350_000, "giáo trình"),
    ("2026-09-10", "chi", "ăn uống", 230_000, "đi chợ"),
    ("2026-09-14", "thu", "làm thêm", 1_200_000, "gia su tuần 2"),
    ("2026-09-15", "chi", "giải trí", 180_000, "xem phim"),
    ("2026-09-18", "chi", "ăn uống", 310_000, "đi chợ"),
    ("2026-09-21", "thu", "làm thêm", 1_200_000, "gia su tuần 3"),
    ("2026-09-25", "chi", "học tập", 99_000, "khóa học trực tuyến"),
]
for ngay, loai, danh_muc, tien, _ in GIAO_DICH:
    if loai == "chi" and tien > 1_000_000:
        print("Khoản chi lớn đầu tiên:", ngay, danh_muc, f"{tien:,}")
        break
```

Kết quả chạy / Output

```
Khoản chi lớn đầu tiên: 2026-09-02 nhà ở 1,800,000
```

Ví dụ "continue": cộng tổng chi bằng cách bỏ qua mọi giao dịch không phải chi. Kết quả 3.134.000 đồng khớp với số liệu của Sổ Xu.

A continue example: add up total spending by skipping every transaction that is not an expense. The result, 3,134,000 dong, matches the Sổ Xu figure.

```
SO_DU_DAU_THANG = 2_500_000
GIAO_DICH = [
    ("2026-09-01", "thu", "gia đình", 4_000_000, "tiền gia đình gửi"),
    ("2026-09-02", "chi", "nhà ở", 1_800_000, "tiền phòng trọ"),
    ("2026-09-03", "chi", "ăn uống", 45_000, "com trưa"),
    ("2026-09-05", "chi", "đi lại", 120_000, "nạp thẻ xe buýt"),
    ("2026-09-07", "thu", "làm thêm", 1_200_000, "gia sư tuần 1"),
    ("2026-09-08", "chi", "học tập", 350_000, "giáo trình"),
    ("2026-09-10", "chi", "ăn uống", 230_000, "đi chợ"),
    ("2026-09-14", "thu", "làm thêm", 1_200_000, "gia sư tuần 2"),
    ("2026-09-15", "chi", "giải trí", 180_000, "xem phim"),
    ("2026-09-18", "chi", "ăn uống", 310_000, "đi chợ"),
    ("2026-09-21", "thu", "làm thêm", 1_200_000, "gia sư tuần 3"),
    ("2026-09-25", "chi", "học tập", 99_000, "khóa học trực tuyến"),
]
tong_chi = 0
for ngay, loai, danh_muc, tien, _ in GIAO_DICH:
    if loai != "chi":
        continue
    tong_chi += tien
print(f"Tổng chi: {tong_chi:,} đồng")
```

Kết quả chạy / Output

```
Tổng chi: 3,134,000 đồng
```

else của vòng lặp là điểm riêng của Python. Khối "else" đặt cùng mức thụt lề với "for" hoặc "while" và chỉ chạy khi vòng lặp kết thúc tự nhiên, tức là **không** gặp "break". Mẫu này hợp với bài toán tìm kiếm: "break" khi thấy, "else" xử lý trường hợp không thấy. Đoạn mã dưới kiểm tra số nguyên tố: nếu có ước trong khoảng từ 2 đến $n - 1$ thì dừng ngay; không có ước nào thì chạy khối "else".

*The **loop else clause** is a feature specific to Python. The else block is at the same indentation as for or while and runs only when the loop ends naturally, that is, **without** meeting break. The pattern suits searching: break when found, and else handles the not-found case. The code below tests for primes: if a divisor exists between 2 and $n - 1$ it stops at once; if there is none, the else block runs.*

```
for n in (29, 21):
    for k in range(2, n):
        if n % k == 0:
            print(n, "không là số nguyên tố, chia hết cho", k)
            break
    else:
        print(n, "là số nguyên tố")
```

Kết quả chạy / Output

```
29 là số nguyên tố
21 không là số nguyên tố, chia hết cho 3
```

Vòng lặp kết thúc bằng <i>The loop ends by</i>	Khối else của vòng lặp <i>The loop else block</i>
Hết dãy (for) hoặc điều kiện sai (while) <i>The sequence ends (for) or the condition turns false (while)</i>	Chạy <i>Runs</i>
Lệnh break <i>A break statement</i>	Không chạy <i>Does not run</i>
Lệnh return trong hàm (chương 5) <i>A return statement in a function (Chapter 5)</i>	Không chạy <i>Does not run</i>

LỖI HAY GẶP / COMMON MISTAKE — Thụt lề "else" sai chỗ: đặt cùng mức với "if" thì "else" thuộc về "if" và chạy ở mọi lần lặp không chia hết, không còn là "else" của vòng lặp. Đây là lỗi logic, chương trình vẫn chạy.

Misplaced else indentation: at the same level as if, the else belongs to the if and runs on every iteration that does not divide, so it is no longer the loop else. This is a logic error; the program still runs.

```
n = 9
for k in range(2, n):
    if n % k == 0:
        print(n, "chia hết cho", k)
        break
    else:
        print(n, "không chia hết cho", k)
```

Kết quả chạy / Output

```
9 không chia hết cho 2
9 chia hết cho 3
```

4.5. Vòng lặp lồng nhau — Nested loops

Vòng lặp lồng là vòng lặp nằm trong thân của vòng lặp khác. Với mỗi lần lặp của vòng ngoài, vòng trong chạy trọn một lượt từ đầu đến cuối. Nếu vòng ngoài lặp 4 lần và vòng trong lặp 3 lần thì thân vòng trong chạy $4 \times 3 = 12$ lần. Vòng lặp lồng dùng cho dữ liệu hai chiều: bảng, hình, lưới.

A nested loop is a loop placed inside the body of another loop. For each iteration of the outer loop, the inner loop runs a complete pass from start to finish. If the outer loop repeats 4 times and the inner loop 3 times, the inner body runs $4 \times 3 = 12$ times. Nested loops serve two-dimensional data: tables, shapes and grids.

Bảng cửu chương 9×9 dùng hai vòng "for": vòng ngoài chọn hàng "i", vòng trong chạy qua cột "j" và in tích "i * j" với độ rộng 4 ký tự. Lệnh "print()" không đối số, thụt ngang hàng với vòng trong, xuống dòng sau mỗi hàng.

*The 9×9 multiplication table uses two for loops: the outer loop picks the row i, the inner loop runs over the column j and prints the product i * j in a width of 4 characters. The argument-free print(), indented level with the inner loop, starts a new line after each row.*

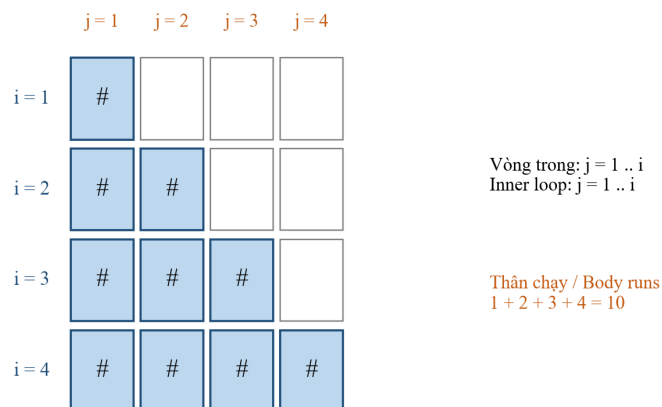
```
for i in range(1, 10):
    for j in range(1, 10):
        print(f"{i * j:4}", end="")
    print()
```

Kết quả chạy / Output

```
1  2  3  4  5  6  7  8  9
2  4  6  8 10 12 14 16 18
3  6  9 12 15 18 21 24 27
4  8 12 16 20 24 28 32 36
5 10 15 20 25 30 35 40 45
6 12 18 24 30 36 42 48 54
7 14 21 28 35 42 49 56 63
8 16 24 32 40 48 56 64 72
9 18 27 36 45 54 63 72 81
```

Hình tam giác bằng ký tự "#": hàng thứ "i" có "i" ký tự. Vòng trong chạy "i" lần và in một ký tự "#" không xuống dòng; sau vòng trong, "print()" xuống dòng. Hình dưới minh họa lưới chỉ số (i, j): ô được tô là các cặp có $j \leq i$.

A triangle of "#" characters: row i has i characters. The inner loop runs i times and prints one "#" without a line break; after the inner loop, print() ends the line. The figure shows the (i, j) index grid: the shaded cells are the pairs with $j \leq i$.



Hình 4.3. Lưới chỉ số của hai vòng lặp lồng khi in tam giác "#"; ô tô màu là các lần in ký tự
Figure 4.3. Index grid of two nested loops when printing a "#" triangle; shaded cells are the times a character is printed

```
n = 5
for i in range(1, n + 1):
    for j in range(i):
        print("#", end="")
    print()
```

Kết quả chạy / Output

```
#
##
###
####
#####
```

Phép nhân chuỗi cho cách ngắn hơn: "#" * i thay cho vòng trong. Khi bài toán cần canh giữa, cộng thêm khoảng trắng đứng trước: hàng "i" của tam giác cân cao 4 hàng có "4 - i" dấu cách và "2 * i - 1" ký tự "#".

*String repetition gives a shorter form: "#" * i replaces the inner loop. When the shape must be centred, spaces are added in front: row i of an isosceles triangle four rows high has 4 - i spaces and 2 * i - 1 "#" characters.*

```
n = 4
for i in range(1, n + 1):
    print(" " * (n - i) + "#" * (2 * i - 1))
```

Kết quả chạy / Output

```
  #
 ###
####
#####
```

"break" trong vòng lồng chỉ thoát vòng trong. Đoạn mã dưới thoát vòng trong khi "j" bằng 1; vòng ngoài vẫn chạy đủ 3 lần. Muốn thoát cả hai vòng cần một biến cờ hoặc đưa vòng lặp vào hàm rồi dùng "return" (chương 5).

break inside a nested loop leaves only the inner loop. The code below leaves the inner loop when j equals 1; the outer loop still runs 3 times. To leave both loops you need a flag variable, or to place the loops in a function and use return (Chapter 5).

```
dem = 0
for i in range(3):
    for j in range(3):
        if j == 1:
            break
        dem += 1
        print(i, j)
print("Số lần in:", dem)
```

Kết quả chạy / Output

```
0 0
1 0
2 0
Số lần in: 3
```

LỖI HAY GẶP / COMMON MISTAKE — Dùng cùng tên biến lặp cho hai vòng lồng. Vòng trong ghi đè biến của vòng ngoài, nên giá trị đọc được sau vòng trong là của vòng trong. Đặt tên khác nhau ("i", "j" hoặc tên có nghĩa như "hang", "cot").

Using the same loop-variable name for two nested loops. The inner loop overwrites the outer variable, so the value read after the inner loop belongs to the inner loop. Use different names (i, j, or meaningful names such as row and column).

```
for i in range(3):
    for i in range(2):
        pass
    print("i ngoài =", i)
```

Kết quả chạy / Output

```
i ngoài = 1
i ngoài = 1
i ngoài = 1
```

4.6. Các mẫu thuật toán với vòng lặp — *Algorithm patterns with loops*

Phần lớn bài toán dùng vòng lặp thuộc năm mẫu: tích lũy (cộng dồn), đếm, tích (nhân dồn), tìm lớn nhất – nhỏ nhất và dựng chuỗi. Mỗi mẫu có giá trị khởi tạo riêng; khởi tạo sai là nguồn lỗi logic phổ biến nhất.

Most problems that use loops fit five patterns: accumulating a sum, counting, accumulating a product, finding the largest or smallest, and building a string. Each pattern has its own starting value; a wrong starting value is the most common source of logic errors.

Mẫu <i>Pattern</i>	Khởi tạo <i>Initial value</i>	Thao tác trong thân <i>Action in the body</i>
Tích lũy tổng <i>Sum</i>	tong = 0	tong += x
Đếm <i>Count</i>	dem = 0	if điều_kiện: dem += 1 <i>if condition: dem += 1</i>
Tích <i>Product</i>	tich = 1	tich *= x
Lớn nhất <i>Largest</i>	Giá trị đầu tiên hoặc None <i>The first value, or None</i>	if x > lon_nhat: lon_nhat = x
Nhỏ nhất <i>Smallest</i>	Giá trị đầu tiên hoặc None <i>The first value, or None</i>	if x < nho_nhat: nho_nhat = x
Dựng chuỗi <i>Build a string</i>	s = ""	s += ky_tu

Một lần duyệt có thể chạy nhiều mẫu cùng lúc. Đoạn mã dưới duyệt 12 giao dịch của Sổ Xu một lần để tính tổng thu, tổng chi, số khoản chi, khoản chi lớn nhất và số dư cuối. Kết quả khớp số liệu chuẩn: thu 7.600.000, chi 3.134.000 (8 khoản), số dư cuối 6.966.000 đồng.

One pass can run several patterns at once. The code below goes through the 12 Sổ Xu transactions once to compute total income, total spending, the number of expenses, the largest expense and the closing balance. The results match the reference figures: income 7,600,000, spending 3,134,000 (8 items), closing balance 6,966,000 dong.

```
SO_DU_DAU_THANG = 2_500_000
GIAO_DICH = [
    ("2026-09-01", "thu", "gia đình", 4_000_000, "tiền gia đình gửi"),
    ("2026-09-02", "chi", "nhà ở", 1_800_000, "tiền phòng trọ"),
    ("2026-09-03", "chi", "ăn uống", 45_000, "com trưa"),
    ("2026-09-05", "chi", "đi lại", 120_000, "nạp thẻ xe buýt"),
    ("2026-09-07", "thu", "làm thêm", 1_200_000, "gia sư tuần 1"),
    ("2026-09-08", "chi", "học tập", 350_000, "giáo trình"),
    ("2026-09-10", "chi", "ăn uống", 230_000, "đi chợ"),
    ("2026-09-14", "thu", "làm thêm", 1_200_000, "gia sư tuần 2"),
    ("2026-09-15", "chi", "giải trí", 180_000, "xem phim"),
    ("2026-09-18", "chi", "ăn uống", 310_000, "đi chợ"),
    ("2026-09-21", "thu", "làm thêm", 1_200_000, "gia sư tuần 3"),
    ("2026-09-25", "chi", "học tập", 99_000, "khóa học trực tuyến"),
]
tong_thu = 0
tong_chi = 0
so_chi = 0
chi_lon = 0
ngay_lon = ""
for ngay, loai, danh_muc, tien, _ in GIAO_DICH:
    if loai == "thu":
        tong_thu += tien
    else:
        tong_chi += tien
        so_chi += 1
        if tien > chi_lon:
            chi_lon = tien
            ngay_lon = ngay
so_du = SO_DU_DAU_THANG + tong_thu - tong_chi
print(f"Tổng thu: {tong_thu:,}")
print(f"Tổng chi: {tong_chi:,} ({so_chi} khoản)")
print(f"Chi lớn nhất: {chi_lon:,} ngày {ngay_lon}")
print(f"Số dư cuối: {so_du:,}")
```

Kết quả chạy / Output

```
Tổng thu: 7,600,000
Tổng chi: 3,134,000 (8 khoản)
Chi lớn nhất: 1,800,000 ngày 2026-09-02
Số dư cuối: 6,966,000
```

Đếm và cộng có điều kiện theo danh mục: ba khoản "ăn uống" (45.000, 230.000 và 310.000) cộng lại 585.000 đồng, đúng bằng chi ăn uống chuẩn của Sổ Xu.

Conditional counting and summing by category: the three "ăn uống" (food) items (45,000, 230,000 and 310,000) add up to 585,000 dong, equal to the reference food spending of Sổ Xu.

```
SO_DU_DAU_THANG = 2_500_000
GIAO_DICH = [
    ("2026-09-01", "thu", "gia đình", 4_000_000, "tiền gia đình gửi"),
    ("2026-09-02", "chi", "nhà ở", 1_800_000, "tiền phòng trọ"),
    ("2026-09-03", "chi", "ăn uống", 45_000, "com trưa"),
    ("2026-09-05", "chi", "đi lại", 120_000, "nạp thẻ xe buýt"),
    ("2026-09-07", "thu", "làm thêm", 1_200_000, "gia sư tuần 1"),
    ("2026-09-08", "chi", "học tập", 350_000, "giáo trình"),
    ("2026-09-10", "chi", "ăn uống", 230_000, "đi chợ"),
    ("2026-09-14", "thu", "làm thêm", 1_200_000, "gia sư tuần 2"),
    ("2026-09-15", "chi", "giải trí", 180_000, "xem phim"),
    ("2026-09-18", "chi", "ăn uống", 310_000, "đi chợ"),
    ("2026-09-21", "thu", "làm thêm", 1_200_000, "gia sư tuần 3"),
    ("2026-09-25", "chi", "học tập", 99_000, "khóa học trực tuyến"),
]
dem = 0
tong = 0
for ngay, loai, danh_muc, tien, _ in GIAO_DICH:
    if loai == "chi" and danh_muc == "ăn uống":
        dem += 1
        tong += tien
print(f"Ăn uống: {dem} khoản, {tong:,} đồng")
```

Kết quả chạy / Output

```
Ăn uống: 3 khoản, 585,000 đồng
```

Tìm nhỏ nhất cần một giá trị khởi đầu không làm sai kết quả. Cách an toàn là đặt "None" rồi gán giá trị đầu tiên gặp được; điều kiện "chi_nho is None or tien < chi_nho" xử lý cả lần đầu lẫn các lần sau.

Finding the smallest needs a starting value that cannot distort the result. The safe way is to set None and then take the first value met; the condition chi_nho is None or tien < chi_nho handles the first pass and the later ones.

```
SO_DU_DAU_THANG = 2_500_000
GIAO_DICH = [
    ("2026-09-01", "thu", "gia đình", 4_000_000, "tiền gia đình gửi"),
    ("2026-09-02", "chi", "nhà ở", 1_800_000, "tiền phòng trọ"),
    ("2026-09-03", "chi", "ăn uống", 45_000, "com trưa"),
    ("2026-09-05", "chi", "đi lại", 120_000, "nạp thẻ xe buýt"),
    ("2026-09-07", "thu", "làm thêm", 1_200_000, "gia sư tuần 1"),
    ("2026-09-08", "chi", "học tập", 350_000, "giáo trình"),
    ("2026-09-10", "chi", "ăn uống", 230_000, "đi chợ"),
    ("2026-09-14", "thu", "làm thêm", 1_200_000, "gia sư tuần 2"),
    ("2026-09-15", "chi", "giải trí", 180_000, "xem phim"),
    ("2026-09-18", "chi", "ăn uống", 310_000, "đi chợ"),
    ("2026-09-21", "thu", "làm thêm", 1_200_000, "gia sư tuần 3"),
    ("2026-09-25", "chi", "học tập", 99_000, "khóa học trực tuyến"),
]
chi_nho = None
ngay_nho = ""
for ngay, loai, danh_muc, tien, _ in GIAO_DICH:
    if loai != "chi":
        continue
    if chi_nho is None or tien < chi_nho:
        chi_nho = tien
        ngay_nho = ngay
print(f"Chi nhỏ nhất: {chi_nho:,} ngày {ngay_nho}")
```

Kết quả chạy / Output

```
Chi nhỏ nhất: 45,000 ngày 2026-09-03
```

```
tich = 1
for k in range(1, 6):
    tich *= k
print("5! =", tich)
```

Kết quả chạy / Output

```
5! = 120
```

LỖI HAY GẶP / COMMON MISTAKE — Khởi tạo giá trị nhỏ nhất bằng 0. Mọi số tiền đều lớn hơn 0 nên điều kiện "tien < nho_nhat" không bao giờ đúng và kết quả luôn là 0. Khởi tạo lớn nhất bằng 0 chỉ an toàn khi dữ liệu chắc chắn không âm.

Starting the smallest value at 0. Every amount is above 0, so the condition tien < nho_nhat is never true and the result is always 0. Starting the largest value at 0 is safe only when the data cannot be negative.

```
chi = [45_000, 120_000, 230_000]
nho_nhat = 0
for tien in chi:
    if tien < nho_nhat:
        nho_nhat = tien
print("Nhỏ nhất:", nho_nhat)
```

Kết quả chạy / Output

```
Nhỏ nhất: 0
```

4.7. Vòng lặp vô hạn và cách tránh — *Infinite loops and how to avoid them*

Vòng lặp vô hạn là vòng lặp mà điều kiện không bao giờ sai nên chương trình không tự kết thúc. Trong terminal của VS Code, nhấn Ctrl+C để ngắt; Python báo "KeyboardInterrupt". Trước khi chạy vòng lặp "while", kiểm tra ba điều: biến trong điều kiện đã được khởi tạo; thân có lệnh làm biến đó thay đổi; và thay đổi theo hướng đưa điều kiện tới sai.

An infinite loop is a loop whose condition never becomes false, so the program does not end by itself. In the VS Code terminal press Ctrl+C to interrupt it; Python reports KeyboardInterrupt. Before running a while loop, check three things: the variable in the condition has been initialised; the body contains a statement that changes it; and the change moves the condition towards false.

Nguyên nhân <i>Cause</i>	Ví dụ <i>Example</i>	Cách tránh <i>How to avoid it</i>
Quên cập nhật biến <i>Forgetting to update the variable</i>	while i <= 3: print(i)	Thêm i += 1 vào thân <i>Add i += 1 to the body</i>
Cập nhật sai hướng <i>Updating in the wrong direction</i>	while i <= 3: i -= 1	Kiểm hướng thay đổi so với điều kiện <i>Check the direction of change against the condition</i>

Nguyên nhân <i>Cause</i>	Ví dụ <i>Example</i>	Cách tránh <i>How to avoid it</i>
Điều kiện luôn đúng <i>A condition that is always true</i>	<code>while 1 < 2:</code>	Viết lại điều kiện hoặc dùng <code>break</code> <i>Rewrite the condition or use break</i>
So sánh số thực bằng <code>==</code> <i>Comparing floats with ==</i>	<code>while x != 1.0: x += 0.1</code>	Dùng <code><</code> hoặc đếm bằng số nguyên <i>Use < or count with integers</i>

Đoạn mã dưới mô phỏng lỗi quên cập nhật "i" nhưng có một bộ đếm cứu hộ "lan" để dừng sau 4 lần. Nếu không có bộ đếm này, vòng lặp in "i = 1" mãi mãi.

The code below simulates forgetting to update i, but a rescue counter lan stops it after 4 passes. Without that counter the loop would print i = 1 forever.

```
i = 1
lan = 0
while i <= 3:
    print("i =", i)
    lan += 1
    if lan == 4:
        print("Dừng cứu hộ sau", lan, "lần")
        break
```

Kết quả chạy / Output

```
i = 1
i = 1
i = 1
i = 1
Dừng cứu hộ sau 4 lần
```

Sai số số thực làm điều kiện `x != 1.0` không bao giờ sai. Cộng 0,1 mười lần cho 0,9999999999999999 chứ không phải 1,0 (chương 2), nên "x" đi qua 1,0 mà vòng lặp không dừng. Bộ đếm cứu hộ `lan < 15` chặn lại.

Floating-point error stops the condition x != 1.0 from ever turning false. Adding 0.1 ten times gives 0.9999999999999999, not 1.0 (Chapter 2), so x passes 1.0 and the loop does not stop. The rescue counter lan < 15 stops it.

```
x = 0.0
lan = 0
while x != 1.0 and lan < 15:
    x += 0.1
    lan += 1
print("Số lần lặp:", lan)
y = 0.0
for _ in range(10):
    y += 0.1
print("y sau 10 lần cộng:", y)
```

Kết quả chạy / Output

```
Số lần lặp: 15
y sau 10 lần cộng: 0.9999999999999999
```

Vòng lặp vô hạn **có chủ ý** là mẫu `while True` kết hợp `break`: điều kiện dừng nằm ở giữa thân, nên chỉ cần một lệnh `input` thay cho hai lệnh như mẫu lính canh ở mục 4.2.

A *deliberate* infinite loop is the while True pattern combined with break: the stopping test sits in the middle of the body, so only one input call is needed instead of the two used by the sentinel pattern in section 4.2.

```
tong = 0
while True:
    tien = int(input("Khoản chi (0 để dừng): "))
    if tien == 0:
        break
    tong += tien
print("Tổng:", tong)
```

Kết quả chạy / Output

```
Khoản chi (0 để dừng): 50000
Khoản chi (0 để dừng): 70000
Khoản chi (0 để dừng): 0
Tổng: 120000
```

PHẢI THUỘC / MEMORISE — Mỗi vòng lặp "while" cần: (1) biến điều khiển đã khởi tạo, (2) lệnh cập nhật trong thân, (3) cập nhật tiến tới sai. "while True" chỉ dừng khi thân có "break" đạt tới được.

Every while loop needs: (1) an initialised control variable, (2) an update statement in the body, (3) an update that moves towards false. Use while True only when the body has a reachable break.

4.8. Sổ Xu bản 0.4: tiết kiệm, trả góp và menu — Sổ Xu version 0.4: savings, instalments and menu

Nguyễn Hoàng Minh thêm ba tính năng vào Sổ Xu, cả ba đều cần vòng lặp. Tính năng thứ nhất trả lời: gửi tiết kiệm 1.000.000 đồng mỗi tháng với lãi 4,8%/năm ghép lãi theo tháng (0,4%/tháng) thì sau bao nhiêu tháng đạt mục tiêu 20.000.000 đồng? Cuối mỗi tháng, số dư được cộng lãi rồi cộng khoản gửi mới: $so_du = so_du * (1 + lai_thang) + TIET_KIEM_MOI_THANG$. Số tháng chưa biết trước nên dùng "while".

*Nguyễn Hoàng Minh adds three features to Sổ Xu, and all three need loops. The first answers: if he saves 1,000,000 dong a month at 4.8% a year compounded monthly (0.4% a month), after how many months does he reach the goal of 20,000,000 dong? At the end of each month the balance receives interest and then the new deposit: $so_du = so_du * (1 + lai_thang) + TIET_KIEM_MOI_THANG$. The number of months is not known in advance, so while is used.*

```
TIET_KIEM_MOI_THANG = 1_000_000
LAI_SUAT_NAM = 0.048
MUC_TIEU = 20_000_000
lai_thang = LAI_SUAT_NAM / 12
so_du = 0.0
thang = 0
while so_du < MUC_TIEU:
    so_du = so_du * (1 + lai_thang) + TIET_KIEM_MOI_THANG
    thang += 1
    if thang % 5 == 0:
        print(f"Tháng {thang:>2}: {so_du:>12,.0f}")
    print(f"Đạt mục tiêu sau {thang} tháng, số dư {so_du:>12,.0f}")
```

Kết quả chạy / Output

```
Tháng 5: 5,040,160
Tháng 10: 10,181,934
Tháng 15: 15,427,368
Tháng 20: 20,778,554
Đạt mục tiêu sau 20 tháng, số dư 20,778,554
```

Sau 20 tháng số dư là 20.778.554 đồng; tháng thứ 19 mới có 19.699.755 đồng, chưa đủ. Tổng tiền gửi 20.000.000 đồng, phần lãi 778.554 đồng.

After 20 months the balance is 20,778,554 dong; month 19 gives only 19,699,755 dong, not enough. The deposits total 20,000,000 dong and the interest is 778,554 dong.

Tính năng thứ hai là lịch trả góp laptop. Giá 18.000.000 đồng, trả trước 6.000.000 đồng nên dư nợ đầu kỳ là 12.000.000 đồng, trả trong 12 tháng. Mỗi kỳ trả gốc đều 1.000.000 đồng cộng lãi 1,5% trên dư nợ còn lại; dư nợ giảm dần nên lãi giảm dần. Số kỳ đã biết, nên dùng "for" với "range".

The second feature is the laptop instalment schedule. The price is 18,000,000 dong with 6,000,000 dong paid up front, so the opening debt is 12,000,000 dong, repaid over 12 months. Each period pays an equal principal of 1,000,000 dong plus 1.5% interest on the remaining debt; the debt falls, so the interest falls. The number of periods is known, so for with range is used.

```
GIA_LAPTOP = 18_000_000
TRA_TRUOC = 6_000_000
KY_HAN_THANG = 12
LAI_SUAT_TRA_GOP_THANG = 0.015
du_no = GIA_LAPTOP - TRA_TRUOC
goc_ky = du_no // KY_HAN_THANG
tong_lai = 0
print(f"{'Kỳ':>3}{ 'Dư nợ':>12}{ 'Gốc':>10}{ 'Lãi':>9}{ 'Phải trả':>11}")
for ky in range(1, KY_HAN_THANG + 1):
    lai = du_no * LAI_SUAT_TRA_GOP_THANG
    phai_tra = goc_ky + lai
    print(f"{'ky':>3}{du_no:>12,}{goc_ky:>10,}{lai:>9,.0f}{phai_tra:>11,.0f}")
    tong_lai += lai
    du_no -= goc_ky
print(f"Tổng lãi: {tong_lai:,.0f}")
```

Kết quả chạy / Output

Kỳ	Dư nợ	Gốc	Lãi	Phải trả
1	12,000,000	1,000,000	180,000	1,180,000
2	11,000,000	1,000,000	165,000	1,165,000
3	10,000,000	1,000,000	150,000	1,150,000
4	9,000,000	1,000,000	135,000	1,135,000
5	8,000,000	1,000,000	120,000	1,120,000
6	7,000,000	1,000,000	105,000	1,105,000
7	6,000,000	1,000,000	90,000	1,090,000
8	5,000,000	1,000,000	75,000	1,075,000
9	4,000,000	1,000,000	60,000	1,060,000
10	3,000,000	1,000,000	45,000	1,045,000
11	2,000,000	1,000,000	30,000	1,030,000
12	1,000,000	1,000,000	15,000	1,015,000
Tổng lãi: 1,170,000				

Kỳ 1 phải trả 1.180.000 đồng (gốc 1.000.000, lãi 180.000); kỳ 12 chỉ còn 1.015.000 đồng (lãi 15.000). Tổng lãi 12 kỳ là 1.170.000 đồng, nên tổng phải trả sau khi vay là 13.170.000 đồng và tổng chi cho chiếc laptop là 19.170.000 đồng gồm cả 6.000.000 đồng trả trước.

Period 1 pays 1,180,000 dong (principal 1,000,000, interest 180,000); period 12 pays only 1,015,000 dong (interest 15,000). Total interest over 12 periods is 1,170,000 dong, so the total repaid on the loan is 13,170,000 dong and the laptop costs 19,170,000 dong including the 6,000,000 dong paid up front.

Tính năng thứ ba là **menu lặp**: chương trình hiển thị các lựa chọn, thực hiện lựa chọn, rồi quay lại menu cho đến khi người dùng chọn thoát. Menu dùng "while True" với "break" ở nhánh thoát; nhánh "else" bắt lựa chọn sai. Dưới đây là dữ liệu nhập mô phỏng: chọn 1, chọn 2, chọn 7 (không có trong menu), rồi chọn 0.

*The third feature is the **repeating menu**: the program shows the choices, carries out one, and returns to the menu until the user chooses to quit. The menu uses while True with break on the quit branch; the else branch catches a wrong choice. The input below is simulated: choose 1, choose 2, choose 7 (not in the menu), then choose 0.*

```
so_du = 6_966_000
muc_tieu = 20_000_000
while True:
    print("--- SỔ XU ---")
    print("1. Xem số dư")
    print("2. Xem mục tiêu tiết kiệm")
    print("0. Thoát")
    chon = input("Chọn: ")
    if chon == "1":
        print(f"Số dư: {so_du:,} đồng")
    elif chon == "2":
        print(f"Mục tiêu: {muc_tieu:,} đồng")
    elif chon == "0":
        print("Tạm biệt!")
        break
    else:
        print("Lựa chọn không hợp lệ:", chon)
```

Kết quả chạy / Output

```
--- SỔ XU ---
1. Xem số dư
2. Xem mục tiêu tiết kiệm
0. Thoát
Chọn: 1
Số dư: 6,966,000 đồng
--- SỔ XU ---
1. Xem số dư
2. Xem mục tiêu tiết kiệm
0. Thoát
Chọn: 2
Mục tiêu: 20,000,000 đồng
--- SỔ XU ---
1. Xem số dư
2. Xem mục tiêu tiết kiệm
0. Thoát
Chọn: 7
Lựa chọn không hợp lệ: 7
--- SỔ XU ---
1. Xem số dư
2. Xem mục tiêu tiết kiệm
0. Thoát
Chọn: 0
Tạm biệt!
```

4.9. Đếm số lần lặp và độ phức tạp $O(n)$, $O(n^2)$ — Counting iterations and $O(n)$, $O(n^2)$ complexity

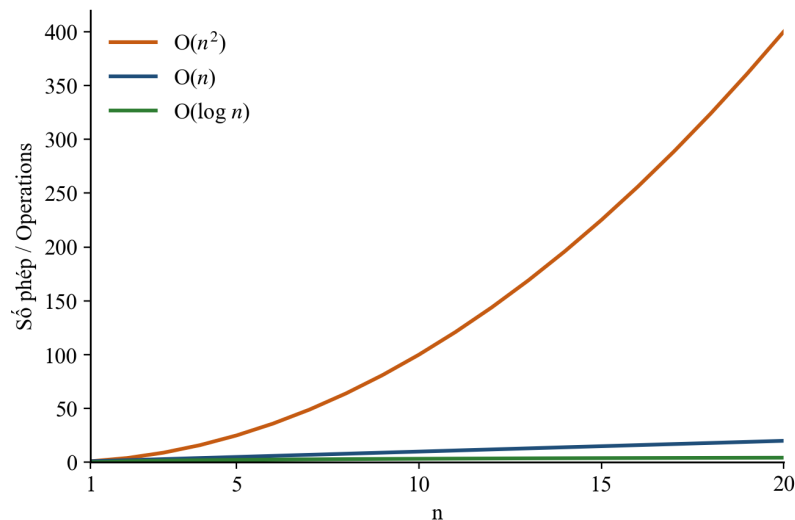
Thời gian chạy của một vòng lặp tỷ lệ với số lần thân của nó được thực hiện. Đếm số lần lặp là cách đo chi phí không phụ thuộc vào máy. **Độ phức tạp thời gian** mô tả số phép tăng thế nào khi kích thước dữ liệu n tăng, và viết bằng ký hiệu O lớn (Big-O), bỏ hằng số và số hạng nhỏ. Một vòng lặp qua n phần tử có độ phức tạp $O(n)$; hai vòng lồng nhau, mỗi vòng chạy n lần, có độ phức tạp $O(n^2)$. Khi n tăng 10 lần thì $O(n)$ tăng 10 lần còn $O(n^2)$ tăng 100 lần.

The running time of a loop is proportional to the number of times its body executes. Counting iterations is a way of measuring cost that does not depend on the machine. Time complexity describes how the number of operations grows as the data size n grows, and is written in Big-O notation, dropping constants and smaller terms. A loop over n items is $O(n)$; two nested loops that each run n times are $O(n^2)$. When n grows tenfold, $O(n)$ grows tenfold while $O(n^2)$ grows a hundredfold.

```
for n in (10, 100, 1000):
    dem1 = 0
    for i in range(n):
        dem1 += 1
    dem2 = 0
    for i in range(n):
        for j in range(n):
            dem2 += 1
    print(f"n={n}>5}  một vòng: {dem1:>5}  hai vòng lồng: {dem2:>9,}")
```

Kết quả chạy / Output

n= 10	một vòng:	10	hai vòng lồng:	100
n= 100	một vòng:	100	hai vòng lồng:	10,000
n= 1000	một vòng:	1000	hai vòng lồng:	1,000,000



Hình 4.4. Số phép theo n của ba mức độ phức tạp: $O(n^2)$ tăng nhanh nhất, $O(\log n)$ chậm nhất
Figure 4.4. Number of operations against n for three complexity classes: $O(n^2)$ grows fastest, $O(\log n)$ slowest

Hai vòng lồng không phải lúc nào cũng chạy n^2 lần. Khi vòng trong bắt đầu từ " $i + 1$ " để xét mỗi cặp đúng một lần, số lần lặp là $n(n - 1)/2$; với $n = 6$ là 15 lần. Độ phức tạp vẫn là $O(n^2)$ vì số hạng lớn nhất là $n^2/2$ và hằng số $1/2$ bị bỏ.

Two nested loops do not always run n^2 times. When the inner loop starts at $i + 1$ so that each pair is visited exactly once, the number of iterations is $n(n - 1)/2$; for $n = 6$ that is 15. The complexity is still $O(n^2)$ because the largest term is $n^2/2$ and the constant $1/2$ is dropped.

```
n = 6
dem = 0
for i in range(n):
    for j in range(i + 1, n):
        dem += 1
print(dem, n * (n - 1) // 2)
```

Kết quả chạy / Output

```
15 15
```

Vòng lặp mà biến điều khiển bị **chia đôi** mỗi lần chỉ chạy khoảng $\log_2 n$ lần, gọi là $O(\log n)$. Với $n = 1.024$ là 10 lần; với $n = 1.000.000$ là 19 lần. Tìm kiếm nhị phân (chương 7) dựa trên chính ý tưởng này.

*A loop whose control variable is **halved** each time runs only about $\log_2 n$ times, called $O(\log n)$. For $n = 1,024$ that is 10 times; for $n = 1,000,000$ it is 19 times. Binary search (Chapter 7) is built on this idea.*

```
for n in (1024, 1_000_000):
    goc = n
    dem = 0
    while n > 1:
        n //= 2
        dem += 1
    print(goc, dem)
```

Kết quả chạy / Output

```
1024 10
1000000 19
```

Lớp Class	Dấu hiệu trong mã Sign in the code	Số phép khi $n = 1.000$ Operations when $n = 1,000$
$O(1)$	Không vòng lặp phụ thuộc n <i>No loop that depends on n</i>	1
$O(\log n)$	Biến điều khiển chia đôi mỗi lần <i>The control variable is halved each time</i>	khoảng 10 <i>about 10</i>
$O(n)$	Một vòng lặp qua n phần tử <i>One loop over n items</i>	1.000 <i>1,000</i>
$O(n^2)$	Hai vòng lồng, mỗi vòng n lần <i>Two nested loops of n passes each</i>	1.000.000 <i>1,000,000</i>

LỖI HAY GẶP / COMMON MISTAKE — Cho rằng thêm một vòng "for" nào cũng làm $O(n)$ thành $O(n^2)$. Hai vòng đứng **nối tiếp** (hết vòng này mới tới vòng kia) vẫn là $O(n)$ vì chỉ cộng $2n$; chỉ vòng **lồng nhau** mới nhân thành n^2 .

*Believing that any extra for loop turns $O(n)$ into $O(n^2)$. Two loops placed **one after the other** are still $O(n)$ because they only add up to $2n$; only **nested** loops multiply into n^2 .*

```
n = 1000
dem = 0
for i in range(n):
    dem += 1
for j in range(n):
    dem += 1
print("Hai vòng nối tiếp:", dem)
```

Kết quả chạy / Output

Hai vòng nối tiếp: 2000

PHẢI THUỘC / MEMORISE — Độ phức tạp thời gian đo bằng số lần thân vòng lặp chạy khi n tăng: một vòng là $O(n)$, hai vòng lồng là $O(n^2)$, vòng chia đôi là $O(\log n)$. Vòng nối tiếp cộng lại, vòng lồng nhân lại.

Time complexity is measured by how many times the loop body runs as n grows: one loop is $O(n)$, two nested loops are $O(n^2)$, a halving loop is $O(\log n)$. Sequential loops add up; nested loops multiply.

4.10. Bảng thuật ngữ — Glossary

Thuật ngữ <i>Term (Vietnamese)</i>	Tiếng Anh <i>English</i>	Giải thích ngắn <i>Short definition</i>
Vòng lặp	loop	Cấu trúc cho một khối lệnh chạy nhiều lần. <i>A structure that runs a block of statements many times.</i>
Lần lặp	iteration	Một lượt chạy hết thân vòng lặp. <i>One pass through the loop body.</i>
Thân vòng lặp	loop body	Khối lệnh thụt lề được lặp lại. <i>The indented block that is repeated.</i>
Vòng lặp while	while loop	Lặp khi điều kiện còn đúng. <i>Repeats while a condition is true.</i>
Vòng lặp for	for loop	Lặp qua từng phần tử của một dãy. <i>Repeats over each item of a sequence.</i>
Biến đếm	counter variable	Biến đếm số lần lặp, tự cập nhật trong thân. <i>A variable that counts iterations, updated inside the body.</i>
Biến lặp	loop variable	Biến nhận giá trị kế tiếp của dãy ở lệnh for. <i>The variable that takes the next value of the sequence in a for statement.</i>
Điều kiện lặp	loop condition	Biểu thức bool quyết định vòng lặp có tiếp tục. <i>The bool expression that decides whether the loop continues.</i>

Thuật ngữ <i>Term (Vietnamese)</i>	Tiếng Anh <i>English</i>	Giải thích ngắn <i>Short definition</i>
Lính canh	sentinel	Giá trị đặc biệt báo hết dữ liệu nhập. <i>A special value that signals the end of input.</i>
Biến cờ	flag	Biến bool ghi nhớ một trạng thái trong vòng lặp. <i>A bool variable that remembers a state during a loop.</i>
Dãy	sequence	Tập phần tử có thứ tự như chuỗi, list, range. <i>An ordered collection such as a string, list or range.</i>
Hàm range	range function	Sinh dãy số nguyên theo start, stop, step. <i>Produces integers from start, stop and step.</i>
Bước nhảy	step	Khoảng cách giữa hai giá trị liên tiếp của range. <i>The gap between two consecutive values of range.</i>
Lệnh break	break statement	Thoát hẳn vòng lặp chứa nó. <i>Leaves the loop that contains it.</i>
Lệnh continue	continue statement	Bỏ qua phần còn lại của lần lặp hiện tại. <i>Skips the rest of the current iteration.</i>
Else của vòng lặp	loop else clause	Khởi chạy khi vòng lặp kết thúc mà không gặp break. <i>A block that runs when the loop ends without meeting break.</i>
Vòng lặp lồng	nested loop	Vòng lặp nằm trong thân vòng lặp khác. <i>A loop inside the body of another loop.</i>
Biến tích lũy	accumulator	Biến cộng dồn hoặc nhân dồn qua các lần lặp. <i>A variable that adds up or multiplies across iterations.</i>
Vòng lặp vô hạn	infinite loop	Vòng lặp mà điều kiện không bao giờ sai. <i>A loop whose condition never becomes false.</i>
Lỗi lệch một	off-by-one error	Lặp thừa hoặc thiếu đúng một lần. <i>Repeating one time too many or too few.</i>
Độ phức tạp thời gian	time complexity	Cách số phép tăng theo kích thước dữ liệu n. <i>How the number of operations grows with the data size n.</i>
Ký hiệu O lớn	Big-O notation	Ký hiệu $O(n)$, $O(n^2)$ mô tả độ phức tạp. <i>Notation such as $O(n)$ or $O(n^2)$ that describes complexity.</i>

4.11. Bài tập lập trình có lời giải — *Worked programming exercises*

Bài 1. Điểm thi với lính canh — *Exercise 1. Exam scores with a sentinel*

Viết chương trình nhập điểm thi của các sinh viên cho đến khi nhập -1 . Điểm hợp lệ nằm trong khoảng từ 0 đến 10; điểm ngoài khoảng bị báo và bỏ qua. In số bài hợp lệ, điểm trung bình (hai chữ số thập phân) và điểm cao nhất. Dữ kiện đề bài: các điểm nhập lần lượt là 7,5; 8; 11; 6,5; 9 rồi -1 .

Write a program that reads students' exam scores until -1 is entered. A valid score lies between 0 and 10; a score outside that range is reported and skipped. Print the number of valid scores, the average (two decimal places) and the highest score. Problem data: the scores entered are 7.5, 8, 11, 6.5, 9 and then -1 .

1. Chưa biết số lần nhập nên dùng "while" với lính canh -1 ; đọc điểm đầu tiên trước vòng lặp, đọc điểm kế tiếp ở cuối thân.
The number of entries is unknown, so use while with the sentinel -1 ; read the first score before the loop and the next score at the end of the body.
2. Ba biến: "tong" (khởi tạo 0), "so_bai" (khởi tạo 0), "cao_nhat" (khởi tạo None).
Three variables: tong (starts at 0), so_bai (starts at 0), cao_nhat (starts at None).
3. Trong thân: điểm ngoài khoảng thì báo; ngược lại cộng vào tổng, tăng đếm, cập nhật điểm cao nhất.
In the body: report a score outside the range; otherwise add it to the total, increase the count and update the highest.
4. Sau vòng lặp, kiểm tra "so_bai > 0" trước khi chia để tránh chia cho 0.
After the loop, test so_bai > 0 before dividing to avoid division by zero.

```
tong = 0.0
so_bai = 0
cao_nhat = None
diem = float(input("Điểm (-1 để dừng): "))
while diem != -1:
    if diem < 0 or diem > 10:
        print("Điểm ngoài khoảng 0 đến 10, bỏ qua")
    else:
        tong += diem
        so_bai += 1
        if cao_nhat is None or diem > cao_nhat:
            cao_nhat = diem
        diem = float(input("Điểm (-1 để dừng): "))
if so_bai > 0:
    print(f"Số bài: {so_bai}")
    print(f"Trung bình: {tong / so_bai:.2f}")
    print(f"Cao nhất: {cao_nhat}")
else:
    print("Chưa có điểm nào")
```