

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>	110
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 and menu</i>	121
4.9. Đếm số lần lặp và độ phức tạp $O(n)$, $O(n^2)$ — <i>Counting iterations and $O(n)$, $O(n^2)$ 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 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 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 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:

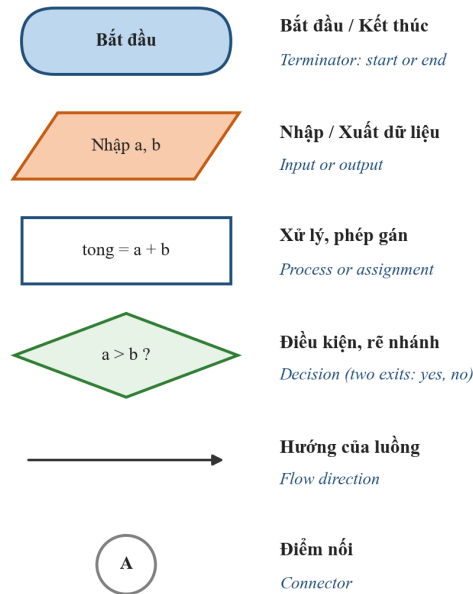
The algorithm for the Sổ Xu end-of-month balance, in words and numbered by step:

- 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

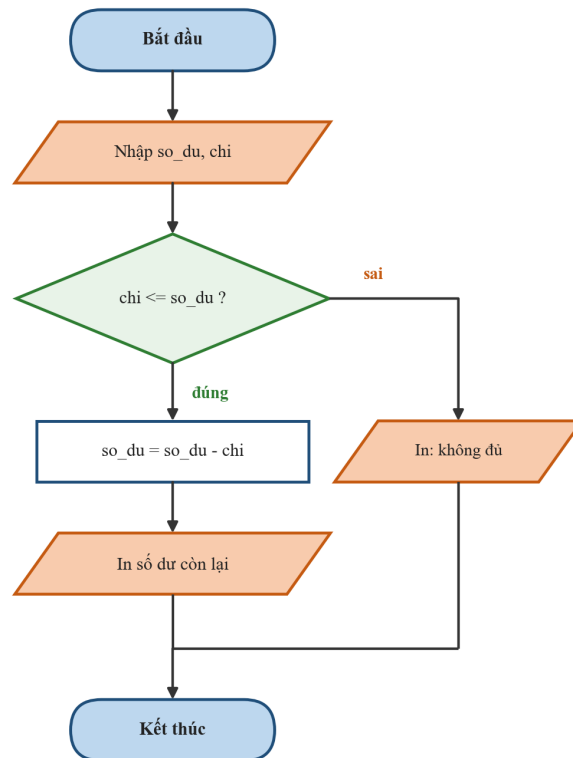
Figure 1.1. Standard flowchart symbols

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.

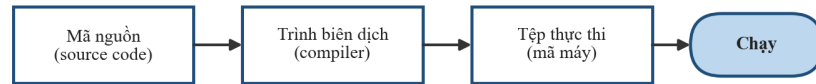
A processor can only carry out instructions in binary form. Programming languages fall into three groups, by how close they are to the machine or to the person.

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.

- 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.
- 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.*
- 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.

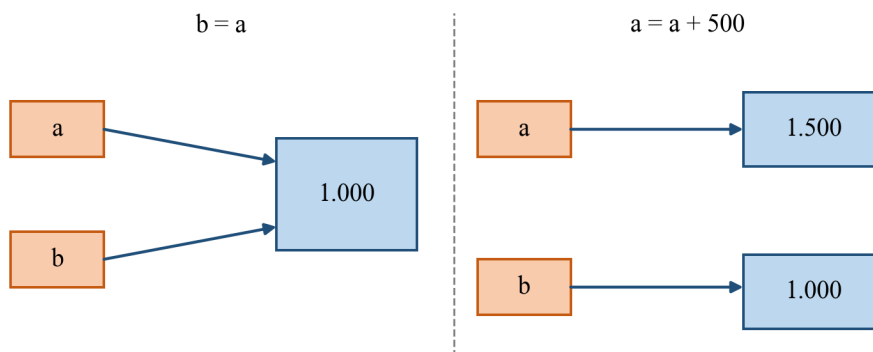
```
a = 1_000
b = a
print(a is b)
a = a + 500
print(a, b, a is b)
```

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 Rule	Đúng Valid	Sai Invalid
Gồm chữ cái, chữ số, dấu gạch dưới Letters, digits and underscores	so_du, tong_chi2	so du, tong-chi
Không bắt đầu bằng chữ số Must not start with a digit	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>	ting_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 <i>Type</i>	Ý nghĩa <i>Meaning</i>	Ví dụ <i>Example</i>	Khả biến <i>Mutable</i>
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>

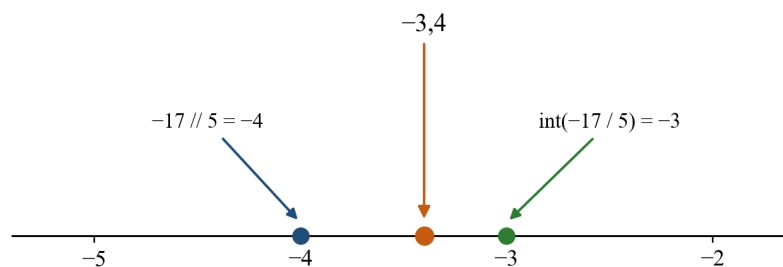
Với số âm, "/" luôn làm tròn **xuống** (về phía $-\infty$) và dấu của "%" theo số chia, sao cho $a == (a // b) * b + a \% b$. Hàm int() thì cắt bỏ phần thập phân (về phía 0), nên "-17 // 5" và "int(-17 / 5)" cho hai kết quả khác nhau.

*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>
<code>math.floor()</code>	xuống <i>down</i>	−4
<code>math.ceil()</code>	lên <i>up</i>	−3
<code>round()</code>	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: +=, -=, *=, /=, //=, %=, **=.*