Chapter 4
Control Flow

Điều khiển luồng trong Python

Python là ngôn ngữ thực thi tuần tự, thay đổi dòng chảy chương trình qua các compound statement như if, for, try, match...

Cấu trúc luôn dùng `:` và **thụt dòng (indentation)** để xác định khối – **không dùng** `{}` **hay** `end`.

`if` / `elif` / `else`: Cấu trúc rẽ nhánh

```
if condition1:
    ...
elif condition2:
    ...
else:
    ...
```

- Mỗi nhánh là một `suite`.
- `elif` là viết tắt của `else` `if`, giúp chuỗi điều kiện rõ ràng, tránh lồng nhau.
- Python đánh giá lần lượt từ trên xuống đến khi gặp điều kiện đúng đầu tiên.

`for` / `while`, `else`: Vòng lặp

```
for item in iterable:
    ...
    if condition:
        break
else:
    # Chạy nếu vòng lặp kết thúc tự nhiên
    ...
```

- `for`: hoạt động qua protocol của **iterator** (`__iter__`, `__next__`).
- `while`: chạy đến khi điều kiện sai.
- `else`: chạy nếu vòng lặp kết thúc tự nhiên mà không `break`.

`try` / `except` / `else` / `finally`: Xử lý lỗi

```
try:
    ...
except SomeError:
    ...
else:
    # Chạy nếu không lỗi
finally:
    # Luôn luôn chạy
```

- Áp dụng nguyên lý EAFP: “Thử làm luôn, nếu lỗi thì xử lý” – trái ngược với kiểm tra trước (LBYL).
- `except` : bắt lỗi cụ thể.
- `else` : tách logic “chạy bình thường” khỏi xử lý lỗi.
- `finally` : đảm bảo tài nguyên được đóng dọn dẹp, kể cả khi có lỗi.

Pattern Matching – `match / case` (Python 3.10+)

```
match data:
    case {"type": "error", "code": x}:
        ...
    case [first, second, *_]:
        ...
    case _:
        ...
```

- **Không chỉ switch-case đơn giản** → So khớp cấu trúc dữ liệu phức tạp: tuple, dict, class...
- **Dừng ở case đầu tiên khớp**.
- `_`: wildcard (mặc định).

Ternary operator – Biểu thức điều kiện inline

```
result = a if cond else b
```

- Là **biểu thức**, không phải câu lệnh, nên có thể dùng được trong context như assignment, hàm.
- Không lồng nhau, dễ đọc.

Các lệnh điều khiển khác

Lệnh	Ý nghĩa
`break`	Dừng vòng lặp ngay lập tức.
`continue`	Bỏ qua phần còn lại, quay lại vòng lặp tiếp theo.
`pass`	Không làm gì – placeholder cho cấu trúc rỗng.
`assert`	Kiểm tra điều kiện, nếu sai thì `AssertionError`.
`return`	Thoát hàm và trả giá trị.
`yield`	Tạo generator, giữ trạng thái giữa các lần gọi.
`raise`	Ném exception (hoặc tái ném lỗi đang bắt).
`del`	Xóa biến, phần tử khỏi list/dict…
`with`	Context manager – quản lý tài nguyên với `__enter__` / `__exit__`.

Python phân tích và thực thi Control Flow như nào?

Bước 1. Phân tích cú pháp → tạo **AST** (abstract syntax tree).

Bước 2. Xác định khối (suite) qua indentation.

Bước 3. Khi gặp câu điều kiện (`if` , `for` , `try` ...), Python tạo **scope logic block**.

Bước 4. Thực thi tuần tự từng block theo control semantics:

Bước 4.1. Đánh giá điều kiện
Bước 4.2. Gọi `__iter__` hoặc `__next__`
Bước 4.3. So khớp `match`
Bước 4.4. Gán handler cho `try`

So sánh nhanh – chọn cái nào khi nào?

Mục đích	Sử dụng chính	Lý do
Chọn nhánh logic	`if / elif / else`	Dễ hiểu, đơn giản, phổ quát.
Duyệt dữ liệu + kiểm tra	`for / while + else`	Tách logic tìm kiếm / xử lý.
Xử lý lỗi	`try / except / else / finally`	Rõ ràng, theo EAFP, đảm bảo cleanup.
Xử lý nhiều nhánh dữ liệu phức	`match / case`	Pattern matching – gọn, dễ đọc, đặc biệt với JSON, dict, tuple.
Viết điều kiện ngắn	`a if cond else b`	Dễ đọc hơn `if` lồng.
Điều khiển vòng lặp	`break`, `continue`, `pass`	Tối ưu logic flow.
Thoát sớm, quản lý tài nguyên	`return`, `yield`, `with`	Rõ ý định logic, tránh rò rỉ tài nguyên.

Kết luận

Python dùng **thụt dòng** và **compound statements** để điều khiển luồng — không cần `{}`. Mỗi cấu trúc mang **triết lý rõ ràng**:

- `if`: chọn lựa.
- `for/while`: lặp logic.
- `try`: xử lý lỗi.
- `match`: biểu diễn cấu trúc.

Biểu thức `ternary`, `with`, `yield`, v.v... giúp viết Python *"pythonic"* hơn.

cheers

cảm ơn

thank you!

muchas gracias

dziękuję

danke