Python Representation Invariants

Table of contents

A **representation invariant** is a condition or property that must remain true about an object's internal state throughout its lifetime. They're primarily a *development-time* tool: they help catch bugs early, simplify method implementations (since you can assume a valid starting state), and document the assumptions a class relies on.

Documenting invariants

State invariants in the class docstring, underneath its attributes:

```
class Fraction():
    """Represents a mathematical fraction.

    Representation Invariants:
    - denominator != 0
    - if fraction is zero, it is represented as 0/1
    """
```

Enforcing invariants

Assertions

```
assert <condition that should be true>, "message if it's not"
```

- Raises `AssertionError` if the condition is false.
- Disabled when Python is run with optimization (`python -O`) — so asserts should catch *bugs*, not perform real input validation that must always run (use `raise ValueError(...)` etc. for that).

The `_check_invariants()` helper pattern

Centralise all invariant checks in one private method, and call it:

- At the end of `__init__`.
- At the end of any method that mutates state.
- At the start of any method that relies on the invariants already holding.

```
def _check_invariants(self) -> None:
    assert self._denom != 0, "Denominator cannot be zero."
    ...
```

Encapsulation: don't expose mutable internals

Returning a direct reference to a private mutable attribute (like a list or dict) lets external code silently violate the class's invariants:

```
def get_members(self):        # BAD -- exposes the real list
    return self._members

def get_members(self):        # GOOD -- caller gets an independent copy
    return list(self._members)
```

Best practices

- Document invariants in docstrings.
- Centralise invariant checking in a single method.
- Don't expose methods/attributes that could let external code violate invariants.
- Write test cases that target edge cases.
- Type hints can express some invariants, but not all (e.g. numeric ranges still need explicit checks).
- Balance strictness with practicality — not every property needs an assertion.