Программирование: как рассчитать и проверить поведение тарифных классов на Python

20.11.2026 · PDF-версия статьи с сайта cms.dolinais.ru

Зачем программисту разбираться в тарифных классах

Тарифные классы и лимиты бронирования — это не только авиационная тема. Это хороший учебный пример системы, где пересекаются **алгоритмы**, **структуры данных** и бизнес-правила. Программирование, по определению Никлауса Вирта, — это «программы = алгоритмы + структуры данных». Здесь мы на Python смоделируем, как авиакомпания закрывает дешёвый тариф, хотя в самолёте есть свободные места, и как это проверить расчётом.

Важно сразу оговориться: мы не претендуем на воспроизведение реальных алгоритмов авиакомпаний. Мы строим **учебную модель**, которая помогает понять логику лимитов и проверить её на данных.

Что такое тарифный класс и лимит бронирования

Тарифный класс (fare class) — это категория билета с определёнными условиями и ценой. **Лимит бронирования** (booking limit) — максимальное число мест, которое можно продать по данному классу на данном рейсе. Даже если физически в самолёте свободно 50 кресел, дешёвый класс может быть закрыт: авиакомпания рассчитывает, что часть мест удастся продать дороже позже.

Ключевая идея: продажа места — это не только «есть кресло или нет», но и **управление доходом** (revenue management). Программисту полезно уметь формализовать такие правила в коде и проверять их на тестовых сценариях.

Модель данных: рейс, классы, спрос

Начнём с простых структур. Нам нужны:

- рейс с общим числом мест;
- набор тарифных классов с ценой и лимитом;
- прогноз спроса по классам на разные горизонты времени.

Опишем это через dataclass — удобный способ задать структуру данных в Python 3.7+ (в 2026 году актуальны версии Python 3.12–3.14, синтаксис dataclass стабилен).

```
from dataclasses import dataclass, field

@dataclass
class FareClass:
    name: str
    price: int
    booking_limit: int

@dataclass
```

```
class Flight:
    code: str
    total_seats: int
    classes: list[FareClass] = field(default_factory=list)
```

Теперь зададим рейс и классы. Цифры условные — они нужны, чтобы показать расчёт, а не чтобы предсказать реальные цены.

```
flight = Flight(
    code="SU-123",
    total_seats=180,
    classes=[
        FareClass("Economy Low", 6000, 40),
        FareClass("Economy Flex", 10000, 80),
        FareClass("Business", 30000, 20),
    ],
)
```

Расчёт доступных мест по классам

Простая проверка: сумма лимитов не должна превышать общее число мест. Если превышает — модель противоречива, и её нужно исправить до расчётов.

```
def validate_limits(flight: Flight) -> None:
    total_limits = sum(c.booking_limit for c in flight.classes)
    if total_limits > flight.total_seats:
        raise ValueError(
            f"????? ??????? {total_limits} ?????? ???? {flight.total_seats}"
        )
```

Дальше считаем, сколько мест останется в каждом классе после гипотетических продаж. Это уже не «истина», а сценарий: мы подставляем прогноз спроса и смотрим, где упрёмся в лимит.

```
def remaining_by_class(flight: Flight, demand: dict[str, int]) -> dict[str, in
t]:
    result = {}
    for c in flight.classes:
        sold = min(demand.get(c.name, 0), c.booking_limit)
        result[c.name] = c.booking_limit - sold
    return result
```

Пример: если спрос на дешёвый класс 60, а лимит 40, продадим 40, остаток 0. Дешёвый тариф «закрыт», хотя в самолёте ещё есть места — именно этот эффект и вызывает удивление у пассажиров.

Как проверить гипотезу на тестовых данных

Хорошая практика — не доверять расчёту «на глаз». Сформулируем проверки:

1. Сумма лимитов не превышает вместимость.
2. Остаток по классу не отрицателен.
3. Если спрос превышает лимит, остаток равен нулю.

Оформим это как простые тесты. В 2026 году стандартный инструмент — pytest, но подойдёт и unittest из стандартной библиотеки.

```
def test_limits():
    validate_limits(flight)
    demand = {"Economy Low": 60, "Economy Flex": 10, "Business": 0}
    left = remaining_by_class(flight, demand)
    assert left["Economy Low"] == 0
    assert left["Economy Flex"] == 70
    assert all(v >= 0 for v in left.values())
```

Такой тест фиксирует поведение модели: при превышении спроса над лимитом класс закрывается. Если правило изменится, тест это покажет.

Где модель ломается и что это значит для программиста

Учебная модель не учитывает отмены, возвраты, пересадки и динамическое изменение цен. В реальности лимиты пересчитываются по мере продаж. Поэтому важно разделять:

- **модель** — упрощение, которое мы проверяем;
- **данные** — фактические продажи и прогнозы;
- **правила** — бизнес-логику, которую нельзя «додумывать» в коде без согласования.

  Если в коде появляется магическое число вроде «40 мест на дешёвый тариф», у него должен быть источник: правило, прогноз или договорённость. Иначе это не модель, а догадка.

Практический вывод: прежде чем писать сложный алгоритм, опишите структуры данных и проверки. Так вы быстрее найдёте противоречия и сможете объяснить результат коллегам — и людям, и ИИ-агентам, которые всё чаще участвуют в проверке кода.

Короткий чек-лист

1. Опишите сущности через dataclass или явные типы.
2. Проверьте, что сумма лимитов не превышает вместимость.

3. Отделите прогноз спроса от правил продажи.
4. Зафиксируйте поведение тестами на граничных значениях.
5. Помните, что модель — упрощение, а не истина.

Такой подход применим не только к авиации: везде, где есть квоты, лимиты и приоритеты, выигрывает тот, кто сначала описывает данные и правила, а потом пишет код.

Источник: https://cms.dolinais.ru/programmirovanie-kak-rasschitat-i-proverit-povedenie-tarifnyh-klassov-na-python