Задание

Тема: тестирование медицинских расчётов и правил валидации данных пациента с помощью `pytest`.

Цель

1. Научиться писать **юнит-тесты** в `pytest` для функций с медицинской логикой.
2. Отработать:
   - параметризацию тестов (`@pytest.mark.parametrize`)
   - проверки исключений (`pytest.raises`)
   - работу с фикстурами (`@pytest.fixture`)
   - граничные случаи и клинически опасные входные данные
3. Понять, как тесты защищают от ошибок в расчётах дозировок и интерпретации показателей.

Задание (шаги)

Шаг 1. Создайте структуру проекта

Сделайте папку, например `med_pytest_hw/`:

- `medcalc.py` — код с функциями
- `test_medcalc.py` — тесты
- (опционально) `requirements.txt` (pytest)

Установка:

```
pip install pytest
```

Запуск тестов:

```
pytest -q
```

Шаг 2. Реализуйте функции в `medcalc.py`

Напишите **ровно эти функции** (названия и сигнатуры важны для тестов):

1. **Расчёт BMI**

```
def bmi(weight_kg: float, height_m: float) -> float:
    """BMI = weight / height^2. Округлить до 2 знаков."""
```

Правила:

- `weight_kg` и `height_m` должны быть > 0
- иначе `ValueError`
- результат округлить `round(x, 2)`

2. **Классификация артериального давления (упрощённо)**

```
def bp_category(systolic: int, diastolic: int) -> str:
    """
    Возвращает одну из категорий:
    'normal', 'elevated', 'stage1', 'stage2', 'crisis'
    """
```

Правила (упрощённая шкала):

- если `systolic < 70` или `diastolic < 40` → `ValueError` (входные данные подозрительны)
- **crisis**: `systolic >= 180` **или** `diastolic >= 120`
- **stage2**: `systolic >= 140` **или** `diastolic >= 90`
- **stage1**: `130–139` **или** `80–89`
- **elevated**: `120–129` и `diastolic < 80`
- **normal**: иначе (то есть `<120` и `<80`)

3. **Проверка совместимости препаратов (игрушечное правило, но полезное для тестов)**

```python
def has_major_interaction(drugs: list[str]) -> bool:
    """
    True если есть противопоказанная комбинация.
    """
```

Правила:

- Нормализовать названия: `strip()`, `lower()`
- Считать взаимодействием комбинации (пара в любом порядке):
  - `("warfarin", "ibuprofen")`
  - `("nitrate", "sildenafil")`
  - `("ssri", "mao inhibitor")`
- Если список пустой → `False`
- Если встречаются пустые строки после `strip()` → `ValueError`

4. **Расчёт дозы по массе (простая модель)**

```python
def dose_mg(weight_kg: float, mg_per_kg: float, max_mg: float) -> float:
    """Доза = weight_kg * mg_per_kg, но не выше max_mg. Округлить до 1 знака."""
```

Правила:

- все входы должны быть > 0, иначе `ValueError`
- `min(weight_kg * mg_per_kg, max_mg)`
- округлить до 1 знака

Шаг 3. Напишите тесты в `test_medcalc.py`

Требования к тестам (минимум):

1. Для `bmi`:
   - тест обычного случая
   - тест округления
   - параметризованный тест для некорректных входов (0, отрицательные)
2. Для `bp_category`:
   - параметризованный тест по категориям (как минимум по 2 примера на категорию)

    - тест на `ValueError` для подозрительно низких значений
3. Для `has_major_interaction`:
    - тест на обнаружение взаимодействий независимо от порядка
    - тест на нормализацию регистра/пробелов
    - тест на пустой список
    - тест на `ValueError` при пустой строке в списке
4. Для `dose_mg`:
    - тест на ограничение `max_mg`
    - тест обычного расчёта
    - тест на `ValueError` при некорректных входах

Шаг 4. Добавьте фикстуру

Сделайте фикстуру в тестах, например:

- список «базовых препаратов без взаимодействий»
- или «типичный пациент» (weight/height/BP)

И используйте её минимум в 2 тестах.

Подсказки по ключевым частям

Что тестировать в мед-логике

- **Границы** (120/129/130/139/140/180 и 80/89/90/120 для АД)
- **Пограничные веса/рост** (например, рост 1.0 м, вес 0.1 кг — должно падать или работать по правилам)
- **Округление** (частая причина скрытых багов)
- **Нормализация строк** (названия препаратов вводятся по-разному)

`pytest.mark.parametrize`

Удобно для таблиц «вход → ожидаемый результат»:

- категории давления
- плохие входы для исключений

- разные комбинации препаратов

`pytest.raises`

Проверяйте не только факт исключения, но и что оно возникает именно там, где нужно:

```
import pytest

with pytest.raises(ValueError):
    bmi(0, 1.7)
```

Про фикстуры

Фикстура — это способ не дублировать подготовку данных:

```
import pytest

@pytest.fixture
def safe_drugs():
    return ["paracetamol", "amoxicillin"]
```

Что проверить перед отправкой (чек-лист)

- [ ] `pytest -q` проходит без ошибок
- [ ] Есть тесты на **нормальные случаи** и **ошибочные входы**
- [ ] Есть **параметризация** хотя бы в 2 местах
- [ ] Есть **фикстура**, используемая минимум в 2 тестах
- [ ] Покрыты **граничные значения** для `bp_category`
- [ ] Проверены **округления** (`bmi` до 2 знаков, `dose_mg` до 1)
- [ ] Проверена **нормализация** строк в `has_major_interaction`
- [ ] Имена функций и файлов совпадают с заданием (`medcalc.py`, `test_medcalc.py`)

Советы по улучшению работы

1. **Думайте как клиницист:** какие ошибки опаснее всего? (например, неверная

категория АД или превышение дозы).

2. **Тестируйте границы, а не “середину”:** большинство багов живёт на порогах.
3. **Разделяйте “валидацию” и “расчёт”:** легче тестировать, меньше скрытых условий.
4. **Пишите тесты так, чтобы они читались как протокол:** входные данные, ожидаемый исход, причина.
5. **Добавьте “негативные” сценарии:** пустые строки, пробелы, неправильный порядок, экстремальные значения.
6. **Держите тесты независимыми:** каждый тест должен проходить в любом порядке.