

## 1. Класс для тестирования:

```
auth_manager.py ×

1  class AuthManager: 10 usages
2      def __init__(self, connection):
3          self.connection = connection
4          self.create_tables()
5
6      def create_tables(self): 1 usage
7          with self.connection:
8              self.connection.execute("""
9                  CREATE TABLE IF NOT EXISTS users (
10                      id INTEGER PRIMARY KEY AUTOINCREMENT,
11                      username TEXT NOT NULL UNIQUE,
12                      password TEXT NOT NULL,
13                      country TEXT NOT NULL,
14                      balance REAL NOT NULL
15                  )
16              """)
17
18      def register_user(self, username, password, country, balance): 25 usages
19          with self.connection:
20              self.connection.execute("""
21                  INSERT INTO users (username, password, country, balance)
22                  VALUES ('{}', '{}', '{}', {})
23                  """.format(*args: username, password, country, balance))
24
25      def authenticate_user(self, username, password): 10 usages
26          cursor = self.connection.cursor()
27          cursor.execute("""
28              SELECT * FROM users
29              WHERE username = '{}' AND password = '{}'
30              """.format(*args: username, password))
31          return cursor.fetchone()
32
33      def delete_user(self, user_id): 1 usage
34          with self.connection:
35              self.connection.execute("""
36                  DELETE FROM users WHERE id = {}
37                  """.format(user_id))
38
39      def get_user_by_id(self, user_id): 14 usages
40          cursor = self.connection.cursor()
41          cursor.execute("""
```

```
auth_manager.py x
1   class AuthManager: 10 usages
33  def delete_user(self, user_id): 1 usage
36      DELETE FROM users WHERE id = {}
37      """.format(user_id))
38
39  def get_user_by_id(self, user_id): 14 usages
40      cursor = self.connection.cursor()
41      cursor.execute("""
42          SELECT * FROM users WHERE id = {}
43          """.format(user_id))
44      return cursor.fetchone()
45
46  def count_users_by_country(self, country): 1 usage
47      cursor = self.connection.cursor()
48      cursor.execute("""
49          SELECT COUNT(*) FROM users WHERE country = '{}'
50          """.format(country))
51      return cursor.fetchone()[0]
52
53  def transfer_balance(self, from_user_id, to_user_id, amount): 3 usages
54      with self.connection:
55          # Проверяем, достаточно ли средств
56          cursor = self.connection.cursor()
57          cursor.execute("SELECT balance FROM users WHERE id = {}".format(from_user_id))
58          from_balance = cursor.fetchone()[0]
59
60          if from_balance < amount:
61              raise ValueError("Insufficient funds")
62          # Выполняем перевод
63          self.connection.execute("""
64              UPDATE users SET balance = balance - {} WHERE id = {}
65              """.format(amount, from_user_id))
66          self.connection.execute("""
67              UPDATE users SET balance = balance + {} WHERE id = {}
68              """.format(amount, to_user_id))
69
```

## 2. Файл базовых тестов:

```
test_base.py ×
1 import pytest
2 import sqlite3
3 from auth_manager import AuthManager
4
5
6 @pytest.fixture 2 usages
7 def db():
8     connection = sqlite3.connect(":memory:")
9     yield connection
10    connection.close()
11
12
13 @pytest.fixture 21 usages
14 def auth_manager(db):
15     return AuthManager(db)
16
17
18 ▶ def test_register_user(auth_manager):
19     auth_manager.register_user( username: "test_user", password: "password123", country: "Russia", balance: 100)
20     user = auth_manager.get_user_by_id(1)
21     assert user is not None, "Пользователь должен быть зарегистрирован"
22     assert user[1] == "test_user", "Имя пользователя должно совпадать"
23     assert user[2] == "password123", "Пароль пользователя должен совпадать"
24     assert user[3] == "Russia", "Страна пользователя должна совпадать"
25     assert user[4] == 100, "Баланс должен быть корректным"
26
27
28 ▶ def test_authenticate_user(auth_manager):
29     auth_manager.register_user( username: "test_user", password: "password123", country: "Russia", balance: 100)
30     user = auth_manager.authenticate_user( username: "test_user", password: "password123")
31     assert user is not None, "Аутентификация должна быть успешной"
32     assert user[1] == "test_user", "Имя пользователя должно совпадать"
33     assert user[2] == "password123", "Пароль должен совпадать"
34
35
36 ▶ def test_get_user_by_id_existing_user(auth_manager):
37     """Тест получения существующего пользователя"""
38     auth_manager.register_user( username: "test_user", password: "password123", country: "Russia", balance: 100)
39     user = auth_manager.get_user_by_id(1)
40     assert user is not None
41     assert user[0] == 1 # ID
```

```

35
36 ▷ def test_get_user_by_id_existing_user(auth_manager):
37     """Тест получения существующего пользователя"""
38     auth_manager.register_user(username="test_user", password="password123", country="Russia", balance=100)
39     user = auth_manager.get_user_by_id(1)
40     assert user is not None
41     assert user[0] == 1 # ID
42
43
44 ▷ def test_get_user_by_id_non_existing_user(auth_manager):
45     """Тест получения несуществующего пользователя"""
46     user = auth_manager.get_user_by_id(999)
47     assert user is None
48
49
50 ▷ def test_get_user_by_id_invalid_input(auth_manager):
51     """Тест с некорректным входным параметром"""
52     with pytest.raises(Exception):
53         auth_manager.get_user_by_id('invalid_id')
54
55
56 ▷ def test_transfer_balance(auth_manager):
57     auth_manager.register_user(username="user1", password="password123", country="CountryA", balance=100)
58     auth_manager.register_user(username="user2", password="password123", country="CountryB", balance=500)
59     user1 = auth_manager.authenticate_user(username="user1", password="password123")
60     user2 = auth_manager.authenticate_user(username="user2", password="password123")
61     try:
62         auth_manager.transfer_balance(user1[0], user2[0], amount=200)
63     except ValueError as e:
64         assert str(
65             e) == "Insufficient funds", "Exception message should be 'Insufficient funds'"
66     update_user1 = auth_manager.get_user_by_id(user1[0])
67     update_user2 = auth_manager.get_user_by_id(user2[0])
68     assert update_user1[4] == 100
69     assert update_user2[4] == 500
70

```

## 2.1. Удачные тесты:

```

(.venv) PS C:\Users\user\PycharmProjects\ практическая_3> pytest test_base.py
=====
platform win32 -- Python 3.12.3, pytest-8.4.2, pluggy-1.6.0
rootdir: C:\Users\user\PycharmProjects\ практическая_3
configfile: pytest.ini
collected 6 items

test_base.py ......

===== 6 passed in 0.00s =====
(.venv) Ps ...

```

## 2.2. Неудачные тесты:

```

=====
platform win32 -- Python 3.12.3, pytest-8.4.2, pluggy-1.6.0
rootdir: C:\Users\user\PycharmProjects\ практическая_3
configfile: pytest.ini
collected 6 items

test_base.py F.....
=====
===== FAILURES =====
----- test_register_user -----
auth_manager = <auth_manager.AuthManager object at 0x000002460A0DBF80>

def test_register_user(auth_manager):
    auth_manager.register_user("test_user", "password123", "Russia", -100)
    user = auth_manager.get_user_by_id(1)
    assert user is not None, "Пользователь должен быть зарегистрирован"
    assert user[1] == "test_user", "Имя пользователя должно совпадать"
    assert user[2] == "password123", "Пароль пользователя должен совпадать"
    assert user[3] == "Russia", "Страна пользователя должна совпадать"
    assert user[4] == -100, "Баланс должен быть корректным"
>E   AssertionError: Баланс должен быть корректным
E   assert -100.0 == 100
E   assert -100.0 == 100

test_base.py:25: AssertionError
=====
===== short test summary info =====
FAILED test_base.py::test_register_user - AssertionError: Баланс должен быть корректным
1 failed, 5 passed in 0.20s =====

```

## 3. Файл параметризованных тестов:

```
test_param.py ×
1 import pytest
2 import sqlite3
3 from auth_manager import AuthManager
4
5
6 @pytest.fixture 2 usages
7 def db():
8     connection = sqlite3.connect(":memory:")
9     yield connection
10    connection.close()
11
12
13 @pytest.fixture 28 usages
14 def auth_manager(db):
15     return AuthManager(db)
16
17
18 @pytest.mark.parametrize("username, password, country, balance",
19                         [ ("user1", "pass1", "CountryA", 100),
20                           ("user2", "pass2", "CountryB", 200),
21                           ("user3", "pass3", "CountryC", 300), ] )
22 ▷ def test_register_user_param(auth_manager, username, password, country, balance):
23     auth_manager.register_user(username, password, country, balance)
24     user = auth_manager.authenticate_user(username, password)
25     assert user is not None, f"Пользователь {username} должен быть зарегистрирован"
26     assert user[1] == username, f"Имя пользователя должно быть {username}"
27     assert user[3] == country, f"Страна пользователя должна быть {country}"
28     assert user[4] == balance, f"Баланс пользователя должен быть {balance}"
29
30
31 @pytest.mark.parametrize("username, password, expected_result", [
32                         ("user1", "password123", True),
33                         ("user2", "wrongpassword", False),
34                         ("nonexistent", "password123", False), ] )
35 ▷ def test_authenticate_user_param(auth_manager, username, password, expected_result):
36     auth_manager.register_user(username="user1", password="password123", country="CountryA", balance=100)
37     auth_manager.register_user(username="user2", password="password456", country="CountryB", balance=200)
38     user = auth_manager.authenticate_user(username, password)
39     if expected_result:
40         assert user is not None, f"Пользователь {username} должен быть аутентифицирован"
```

```
test_param.py ×

35     def test_authenticate_user_param(auth_manager, username, password, expected_result):
36         user = auth_manager.authenticate_user(username, password)
37         if expected_result:
38             assert user is not None, f"Пользователь {username} должен быть аутентифицирован"
39         else:
40             assert user is None, f"Пользователь {username} не должен быть аутентифицирован"
41
42
43
44     @pytest.mark.parametrize("from_balance, to_balance, transfer_amount, expected_from_balance, expected_to_balance",
45                             [ (100, 50, 50, 100),
46                               (200, 100, 150, 50, 250),
47                               (100, 100, 200, 100, 100), ] )
48
49 ▶ def test_transfer_balance_param(auth_manager, from_balance, to_balance, transfer_amount, expected_from_balance, expected_to_balance):
50     # Регистрация пользователей
51     auth_manager.register_user(username: "user1", password: "password123", country: "CountryA", from_balance)
52     auth_manager.register_user(username: "user2", password: "password123", country: "CountryA", to_balance)
53     from_user_id = auth_manager.authenticate_user(username: "user1", password: "password123")[0]
54     to_user_id = auth_manager.authenticate_user(username: "user2", password: "password123")[0]
55     if from_balance >= transfer_amount:
56         auth_manager.transfer_balance(from_user_id, to_user_id, transfer_amount)
57         from_user = auth_manager.get_user_by_id(from_user_id)
58         to_user = auth_manager.get_user_by_id(to_user_id)
59         assert from_user[4] == expected_from_balance, f"У отправителя должно остаться {expected_from_balance} единиц"
60         assert to_user[4] == expected_to_balance, f"У получателя должно быть {expected_to_balance} единиц"
61
62
63     @pytest.mark.parametrize("country, expected_count", [ ("CountryA", 2), ("CountryB", 1), ("CountryC", 0) ])
64 ▶ def test_count_users_by_country(auth_manager, country, expected_count):
65     auth_manager.register_user(username: "user1", password: "password123", country: "CountryA", balance: 1000)
66     auth_manager.register_user(username: "user2", password: "password123", country: "CountryA", balance: 1000)
67     auth_manager.register_user(username: "user3", password: "password123", country: "CountryB", balance: 1000)
68     count = auth_manager.count_users_by_country(country)
69     assert count == expected_count
70
71
72     @pytest.mark.parametrize("user_id, expected", [
73         (1, ("user1", "password123", "CountryA", 1000)),
74         (2, ("user2", "password123", "CountryB", 1000)),
75         (3, None), # Несуществующий пользователь
76         (0, None), # Несуществующий пользователь
77     ])
78
79
80
```

```

61
62
63     @pytest.mark.parametrize("country, expected_count", [("CountryA", 2), ("CountryB", 1), ("CountryC", 0)])
64     def test_count_users_by_country(auth_manager, country, expected_count):
65         auth_manager.register_user(username="user1", password="password123", country="CountryA", balance=1000)
66         auth_manager.register_user(username="user2", password="password123", country="CountryA", balance=1000)
67         auth_manager.register_user(username="user3", password="password123", country="CountryB", balance=1000)
68         count = auth_manager.count_users_by_country(country)
69         assert count == expected_count
70
71
72     @pytest.mark.parametrize("user_id, expected", [
73         (1, (1, "user1", "password123", "CountryA", 1000)),
74         (2, (2, "user2", "password123", "CountryB", 1000)),
75         (3, None), # Несуществующий пользователь
76         (0, None), # Несуществующий пользователь
77         (-1, None), # Несуществующий пользователь
78     ])
79     def test_get_user_by_id(auth_manager, user_id, expected):
80         auth_manager.register_user(username="user1", password="password123", country="CountryA", balance=1000)
81         auth_manager.register_user(username="user2", password="password123", country="CountryB", balance=1000)
82         user = auth_manager.get_user_by_id(user_id)
83         assert user == expected
84
85
86     @pytest.mark.parametrize("user_id, expected", [
87         (1, None),
88         (2, None),
89     ])
90     def test_delete_user(auth_manager, user_id, expected):
91         auth_manager.register_user(username="user1", password="password123", country="CountryA", balance=1000)
92         auth_manager.register_user(username="user2", password="password123", country="CountryB", balance=1000)
93         user = auth_manager.delete_user(user_id)
94         assert user == expected
95

```

### 3.1. Удачные тесты:

```

(.venv) PS C:\Users\user\PycharmProjects\ практическая_3> pytest test_param.py
=====
platform win32 -- Python 3.12.3, pytest-6.4.2, pluggy-1.0.0
rootdir: C:\Users\user\PycharmProjects\ практическая_3
configfile: pytest.ini
collected 19 items

test_param.py ......

===== 19 passed in 0.20s =====
[100%]

```

### 3.2. Неудачные тесты:

```

(.venv) PS C:\Users\user\PycharmProjects\ практическая_3> pytest test_param.py
=====
platform win32 -- Python 3.12.3, pytest-6.4.2, pluggy-1.0.0
rootdir: C:\Users\user\PycharmProjects\ практическая_3
configfile: pytest.ini
collected 19 items

test_param.py ..F.....
=====
***** FAILURES *****
test_get_user_by_id[2-expectd] ..F.....
auth_manager = auth_manager.AuthManager object at 0x0000021826400220>, user_id = 2, expected = (2, 'user2', 'password123', 'CountryB', 1000)

@ pytest.mark.parametrize("user_id, expected", [
    (1, (1, "user1", "password123", "CountryA", 1000)),
    (2, "user2", "password123", "CountryB", 1000),
    (3, None), # Несуществующий пользователь
    (0, None), # Несуществующий пользователь
    (-1, None), # Несуществующий пользователь
])
def test_get_user_by_id(auth_manager, user_id, expected):
    auth_manager.register_user("user1", "password123", "CountryA", 1000)
    auth_manager.register_user("user2", "password123", "CountryB", -1000)
    user = auth_manager.get_user_by_id(user_id)
>     assert user == expected
E     AssertionError: assert (2, 'user2', ...ry8, -1000.0) == (2, 'user2', ...untry8, 1000)
E
E     At index 4 diff: -1000.0 != 1000
E     Use -v to get more diff

test_param.py:83: AssertionError
=====
FAILED test_param.py::test_get_user_by_id[2-expectd] - AssertionError: assert (2, 'user2', ...ry8, -1000.0) == (2, 'user2', ...untry8, 1000)
=====
===== short test summary info =====
1 failed, 19 passed in 0.30s =====

```

### 4. Файл тестирования исключений:

```
test_except.py ×

1 import pytest
2 import sqlite3
3 from auth_manager import AuthManager
4
5
6 @pytest.fixture 2 usages
7 def db():
8     connection = sqlite3.connect(":memory:")
9     yield connection
10    connection.close()
11
12
13 @pytest.fixture 12 usages
14 def auth_manager(db):
15     return AuthManager(db)
16
17
18 @pytest.mark.exception
19 def test_user_not_found(auth_manager):
20     non_existent_user_id = 999
21     user = auth_manager.get_user_by_id(non_existent_user_id)
22     assert user is None
23
24
25 @pytest.mark.exception
26 def test_transfer_insufficient_funds(auth_manager):
27     auth_manager.register_user(username="user1", password="password123", country="CountryA", balance=100)
28     auth_manager.register_user(username="user2", password="password123", country="CountryA", balance=100)
29     from_user_id = auth_manager.authenticate_user(username="user1", password="password123")[0]
30     to_user_id = auth_manager.authenticate_user(username="user2", password="password123")[0]
31     with pytest.raises(ValueError, match="Insufficient funds"):
32         auth_manager.transfer_balance(from_user_id, to_user_id, amount=200)
33
34
35 @pytest.mark.exception
36 def test_dict_type_raises_exception(auth_manager):
37     """Тест передачи словаря"""
38     service = auth_manager
39     with pytest.raises((TypeError, sqlite3.Error)):
40         service.get_user_by_id({"id": 1})
41
42
43 @pytest.mark.exception
44 def test_list_type_raises_exception(auth_manager):
45     """Тест передачи списка"""
46     service = auth_manager
47     with pytest.raises((TypeError, sqlite3.Error)):
48         service.get_user_by_id([1, 2, 3])
49
```

#### 4.1. Удачные тесты:

```
(venv) PS C:\...\ практическая\> pytest test_except.py
=====
platform win32 -- Python 3.12.3, pytest-8.4.2, pluggy-1.8.0
rootdir: C:\...\ практическая\...
configfile: pytest.ini
collected 4 items

test_except.py ....
=====
4 passed in 0.1s [100%]
```

#### 4.2. Неудачные тесты:

```
(.venv) PS C:\...> pytest test_except.py
=====
platform win32 -- Python 3.12.3, pytest-6.0.0
rootdir: C:\...
configfile: pytest.ini
collected 4 items

test_except.py F..

=====
test_transfer_insufficient_funds [100%]

auth_manager = <auth_manager.AuthManager object at 0x000002744283E50>

    @pytest.mark.exception
    def test_transfer_insufficient_funds(auth_manager):
        auth_manager.register_user("user1", "password123", "CountryA", 100)
        auth_manager.register_user("user2", "password123", "CountryA", 100)
        from_user_id = auth_manager.authenticate_user("user1", "password123")[0]
        to_user_id = auth_manager.authenticate_user("user2", "password123")[0]
        with pytest.raises(ValueError, match="Insufficient funds"):
            E = auth_manager.transfer("user1", "user2", 1000)
        Failed: DID NOT RAISE <class 'ValueError'>
test_except.py:31: Failed
=====
FAILED test_except.py::test_transfer_insufficient_funds - Failed: DID NOT RAISE <class 'ValueError'>
=====
short test summary info =====
1 failed, 3 passed in 0.28s =====
```

## 5. Файл тестов с использованием методов:

```
test_mark.py

1 import pytest
2 import sqlite3
3 from auth_manager import AuthManager
4
5
6 @pytest.fixture 2 usages
7 def db():
8     connection = sqlite3.connect(":memory:")
9     yield connection
10    connection.close()
11
12
13 @pytest.fixture 15 usages
14 def auth_manager(db):
15     return AuthManager(db)
16
17
18 @pytest.mark.exception
19 def test_register_user_with_existing_username(auth_manager):
20     auth_manager.register_user(username="user1", password="password123", country="CountryA", balance=100)
21     with pytest.raises(sqlite3.IntegrityError, match="UNIQUE constraint failed: users.username"):
22         auth_manager.register_user(username="user1", password="password456", country="CountryB", balance=200)
23
24
25 @pytest.mark.exception
26 def test_authenticate_user_with_wrong_password(auth_manager):
27     auth_manager.register_user(username="user1", password="password123", country="CountryA", balance=100)
28     user = auth_manager.authenticate_user(username="user1", password="wrongpassword")
29     assert user is None, "Аутентификация должна вернуть None для неверного пароля"
30
31
32 @pytest.mark.basic
33 def test_create_tables(auth_manager):
34     """Тест создания таблиц"""
35     # Проверяем что таблица создалась
36     cursor = auth_manager.connection.cursor()
37     cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND name='users'")
38     result = cursor.fetchone()
39     assert result is not None
40     assert result[0] == 'users'
```

```

test_mark.py x
33     def test_create_tables(auth_manager):
34         cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND name='users'")
35         result = cursor.fetchone()
36         assert result is not None
37         assert result[0] == 'users'
38
39
40
41
42
43     @pytest.mark.basic
44     def test_register_user_success(auth_manager):
45         """Тест успешной регистрации пользователя"""
46         # Регистрируем пользователя
47         auth_manager.register_user(username="testuser", password="password123", country="Russia", balance=100.0)
48
49         # Проверяем что пользователь добавился
50         user = auth_manager.get_user_by_id(1)
51         assert user is not None
52         assert user[1] == "testuser" # username
53         assert user[2] == "password123" # password
54         assert user[3] == "Russia" # country
55         assert user[4] == 100.0 # balance
56
57
58     @pytest.mark.skip(reason="Функция верификации пароля еще не реализована")
59     def test_password_strength_verification(self, auth_manager):
60         """Тест проверки сложности пароля - НЕ РЕАЛИЗОВАНО"""
61         # Этот тест будет пропущен, так как функциональность еще не готова
62         with pytest.raises(ValueError, match="Password is too weak"):
63             auth_manager.register_user(username="user1", password="123", country="Country", balance=100.0)
64
65
66     @pytest.mark.skip(reason="API для проверки страны временно недоступно")
67     def test_country_validation(self, auth_manager):
68         """Тест валидации страны"""
69         with pytest.raises(ValueError, match="Invalid country"):
70             auth_manager.register_user(username="user1", password="password", country="NonexistentCountry", balance=100.0)
71

```

## 5.1. Удачные тесты:

```

(.venv) PS C:\...\ практическая_3> pytest test_mark.py
=====
platform win32 -- Python 3.12.3, pytest-8.4.2, pluggy-1.6.0
rootdir: C:\...\ практическая_3\ практическая_3
configfile: pytest.ini
collected 6 items

test_mark.py ...ss

===== 4 passed, 2 skipped in 0.05s =====
[100%]

```

## 5.2. Неудачные тесты:

```

(.venv) PS C:\...\ практическая_3> pytest test_mark.py
=====
platform win32 -- Python 3.12.3, pytest-8.4.2, pluggy-1.6.0
rootdir: C:\...\ практическая_3\ практическая_3
configfile: pytest.ini
collected 6 items

test_mark.py ...Fss

===== FAILURES =====
=====
test_register_user_success
=====
auth_manager = <auth_manager.AuthManager object at 0x00000018F405C0B0>

@pytest.mark.basic
def test_register_user_success(auth_manager):
    """Тест успешной регистрации пользователя"""
    # Регистрируем пользователя
    auth_manager.register_user("testuser", "password123", "Russia", 100.0)

    # Проверяем что пользователь добавился
    user = auth_manager.get_user_by_id(1)
    assert user is not None
    assert user[1] == "testuser" # username
    assert user[2] == "password123" # password
    assert user[3] == "Russia" # country
    assert user[4] == 100.0 # balance
    ^^^^^^^^^^^^^^^^^^
    E   assert 100.0 == -100.0

test_mark.py:55: AssertionError
=====
short test summary info
=====
FAILED test_mark.py::test_register_user_success - assert 100.0 == -100.0
=====
1 failed, 3 passed, 2 skipped in 0.26s =====
[100%]

```

## 6. Файл теста на SQL-инъекцию:

```
test_injection.py
1 import pytest
2 import sqlite3
3 from auth_manager import AuthManager
4
5
6 @pytest.fixture 2 usages
7 def db():
8     connection = sqlite3.connect(":memory:")
9     yield connection
10    connection.close()
11
12
13 @pytest.fixture 2 usages
14 def auth_manager(db):
15     return AuthManager(db)
16
17
18 @pytest.mark.exception
19 def test_sql_injection_drop_table(auth_manager):
20     """Тест SQL инъекции с DROP TABLE"""
21     with pytest.raises(sqlite3.Error) as exc_info:
22         auth_manager.get_user_by_id("DROP TABLE users")
23     # Проверяем что это ошибка SQL синтаксиса|
24     error_message = str(exc_info.value).lower()
25     assert any(word in error_message for word in ['syntax', 'near', 'error'])
26
```

### 6.1. Удачный тест:

```
(venv) PS C:\Users\...\\ практика_3> pytest test_injection.py
=====
platform win32 -- Python 3.12.3, pytest-8.4.2, pluggy-1.0.0
rootdir: C:\Users\...\ практика_3
configfile: pytest.ini
collected 1 item
test_injection.py .
=====
1 passed in 0.08s [100%]
(venv) PS C:\Users\...\\ практика_3> []
```