# Task 1

**Description:**Provide the Python code based on the [[fixture\_task.py](https://github.com/AntonLazarchik/DQE_LAB_2024_CODE_SAMPLES/blob/main/fixture_task.py)](https://github.com/AntonLazarchik/DQE_LAB_2024_CODE_SAMPLES/blob/main/fixture_task.py) which will contain 2 fixtures. The first fixture is used to track the time of the test execution. The second will track the time of the whole suite execution. All tests must use fixture 1 except the last one. The fixture for the suite time execution must be invoked once.

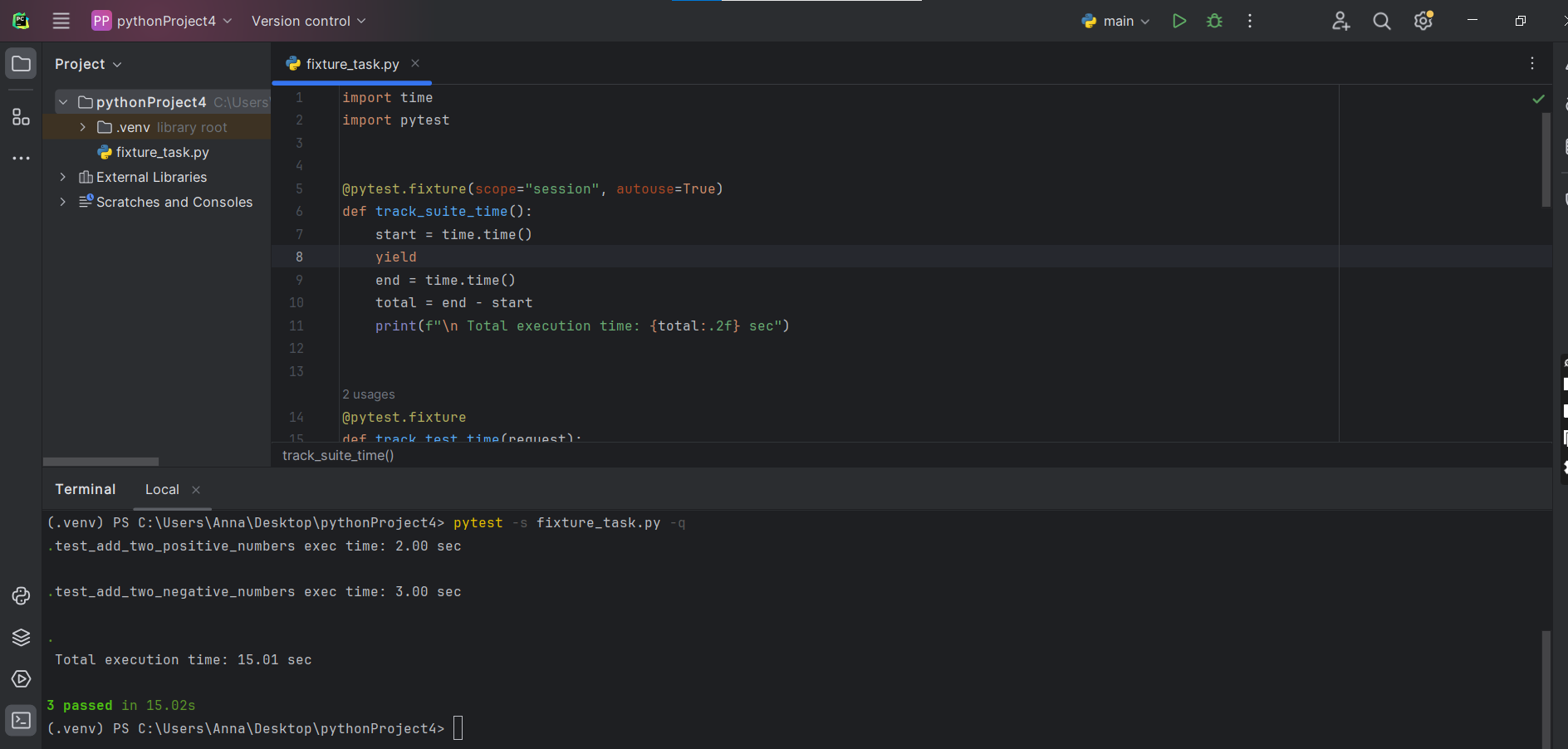
What is expected to be used in the task: yield in fixtures, fixture scopes, autouse flag, only time module.

**Expected result**: The Python file with implemented code. The screenshot with run results.

fixture\_task.py:

import time  
import pytest  
  
  
@pytest.fixture(scope="session", autouse=True)  
def track\_suite\_time():  
 start = time.time()  
 yield  
 end = time.time()  
 total = end - start  
 print(f"\n Total execution time: {total:.2f} sec")  
  
  
@pytest.fixture  
def track\_test\_time(request):  
 start = time.time()  
 yield  
 end = time.time()  
 duration = end - start  
 print(f"{request.node.name} exec time: {duration:.2f} sec\n")  
  
  
def add\_numbers(a, b):  
 return a + b  
  
  
def test\_add\_two\_positive\_numbers(track\_test\_time):  
 a, b = 3, 5  
 result = add\_numbers(a, b)  
 time.sleep(2)  
 assert result == 8  
  
  
def test\_add\_two\_negative\_numbers(track\_test\_time):  
 a, b = -3, -5  
 result = add\_numbers(a, b)  
 time.sleep(3)  
 assert result == -8  
  
  
def test\_add\_negative\_and\_positive\_numbers():  
 a, b = -3, 5  
 result = add\_numbers(a, b)  
 time.sleep(10)  
 assert result == 2

Screenshot:



# Task 2

**Description:**

You need to write the logic for the tests and use parametrisation for the test\_add\_numbers function based on the code [parametrize\_task.py](https://github.com/AntonLazarchik/DQE_LAB_2024_CODE_SAMPLES/blob/main/parametrize_task.py). The names of the cases must be also used from the config file [config.yaml](https://github.com/AntonLazarchik/DQE_LAB_2024_CODE_SAMPLES/blob/main/config.yaml).

For test\_add\_invalid\_types you need to verify that the specific exception type is returned. Also, mark the test\_add\_numbers as smoke and test\_add\_invalid\_types as critical.

Launch tests 3 times using the flags in pytest to run only smoke and second time only critical path tests.

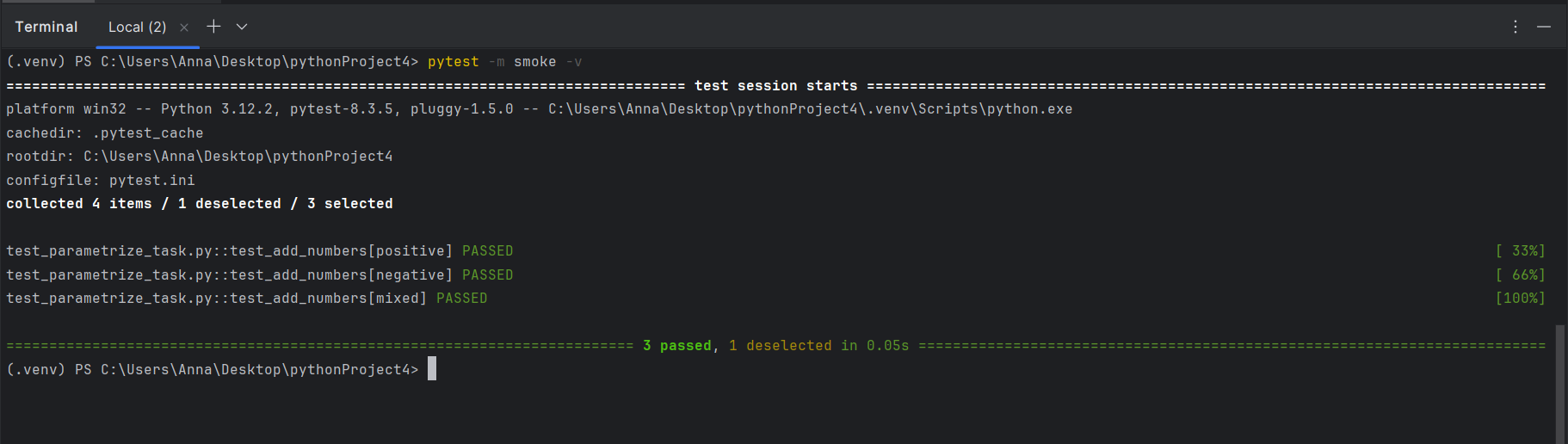
**What is expected to be used in the task:** @pytest.mark, pytest.raises, assert, pytest run commands and flags.

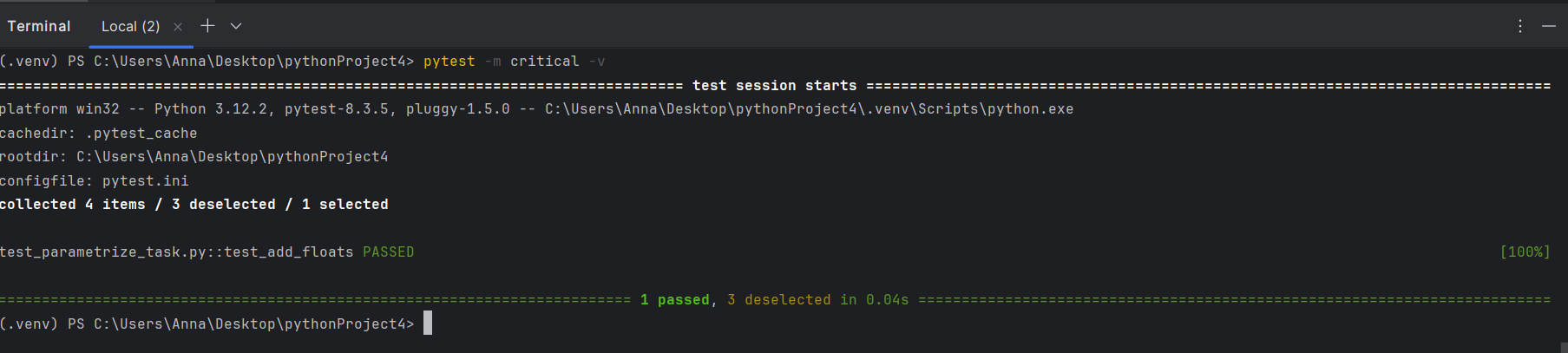
**Expected result:** The Python file with implemented code. The screenshot with 3 run results. First run only for tests marked as smoke. Second run for critical path tests. The third run for the whole suite.

parametrize\_task.py:

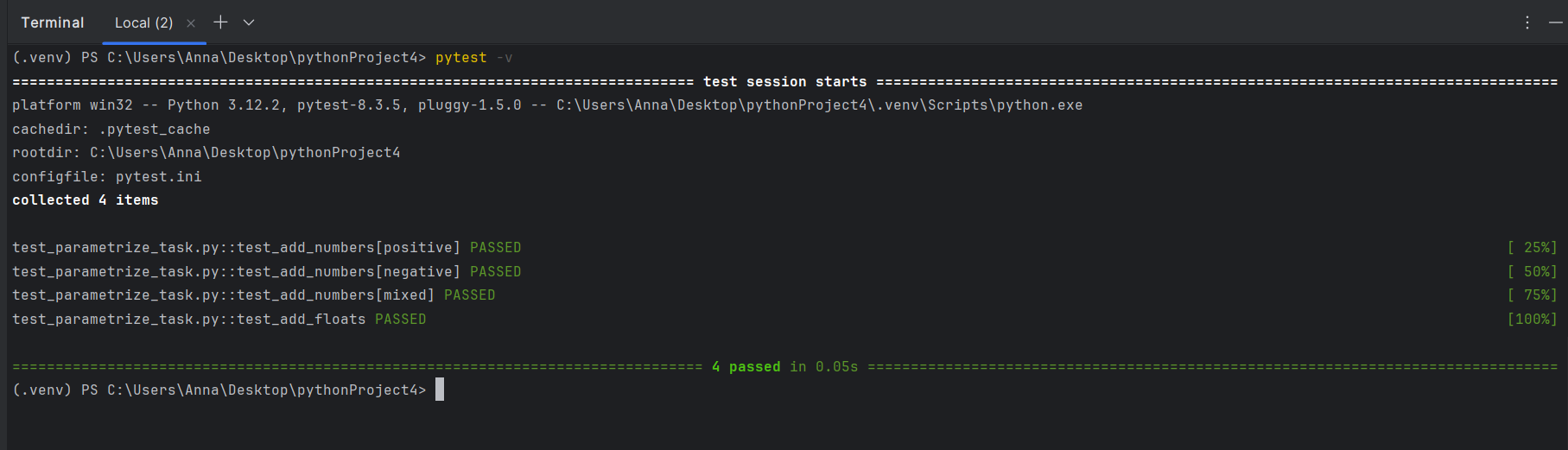
import pytest  
import yaml  
  
  
def get\_numbers\_data(config\_name):  
 with open(config\_name, 'r') as stream:  
 config = yaml.safe\_load(stream)  
 return config['cases']  
  
  
def add\_numbers(a, b, c):  
 try:  
 return a + b + c  
 except TypeError:  
 raise TypeError('Please check the parameters. All of them must be numeric')  
  
  
test\_data = get\_numbers\_data("config.yaml")  
params = [tuple(case["input"]) + (case["expected"],) for case in test\_data]  
ids = [case["case\_name"] for case in test\_data]  
  
  
@pytest.mark.smoke  
@pytest.mark.parametrize("a, b, c, expected", params, ids=ids)  
def test\_add\_numbers(a, b, c, expected):  
 result = add\_numbers(a, b, c)  
 assert result == expected  
  
  
@pytest.mark.critical  
def test\_add\_floats():  
 a, b, c = 'a', 2, 1  
 with pytest.raises(TypeError):  
 add\_numbers(a, b, c)

Screenshots:  
First run:



Second run:  


Third run:



# Task 3

**General Overview:** You need to implement the basic framework to test the tables in your database.

1. 6 tests must be implemented: 3 for smoke tests like the presence of the objects. 3 for the critical path. Any tests can be used for smoke or critical path tests. The main requirement is to have the possibility to run smoke or critical path tests in the same suite or separately.

2. The result of the test run must be provided using the allure. 1 Report for smoke tests. 1 report for critical path. 1 for the whole suite (smoke + critical). Use the ‘–single-file’ flag.

3. Your tests must have custom marks for smoke and critical paths.

4. The SQL queries must be passed through ‘parametrize’ functionality and stored in config files. Config structure example: [config\_SQL\_example.yaml](https://github.com/AntonLazarchik/DQE_LAB_2024_CODE_SAMPLES/blob/main/config_SQL_example.yaml)

5. The connection to the DB must be implemented and passed to the tests through a fixture. The connection must be closed as a teardown step.

6. conftest.py must be used to store the global configs and fixtures.

7\*. Generate the allure report during the test run. (pytest\_sessionfinish and subprocess)

The connection to the DB could be established using the pyodbc module for SQL Server or the psycopg2 module for Postgres. The configuration to connect to the DB can be found in the code samples [useful\_stuff.py](https://github.com/AntonLazarchik/DQE_LAB_2024_CODE_SAMPLES/blob/main/useful_stuff.py).

**What is expected to be used in the task:** @pytest.mark, assert, allure commands, allure.step, conftest, pytest.ini, pytest run commands and flags.

**Expected result:** The Python files with implemented code. 3 Allure reports (First run only for tests marked as smoke. Second run for critical path tests. The third run for the whole suite)

pytest.ini:

[pytest]  
markers =  
 smoke: mark test as a smoke test  
 critical: mark test as a critical-path test  
  
addopts =  
 --alluredir=reports/allure\_raw  
 --clean-alluredir

conftest.py:

import yaml  
import pytest  
import psycopg2  
import subprocess  
  
def pytest\_addoption(parser):  
 parser.addoption('--db-host', default='localhost', help='Database host')  
 parser.addoption('--db-port', default='5432', help='Database port')  
 parser.addoption('--db-name', default='test\_EPAM', help='Database name')  
 parser.addoption('--db-user', default='postgres', help='Database user')  
 parser.addoption('--db-pass', default='password', help='Database password')  
  
def pytest\_generate\_tests(metafunc):  
 if 'test\_def' not in metafunc.fixturenames:  
 return  
 with open('config\_SQL\_example.yaml') as f:  
 tests = yaml.safe\_load(f)['tests']  
  
 params = []  
 for t in tests:  
 marks = []  
 if t.get('mark') == 'smoke':  
 marks.append(pytest.mark.smoke)  
 elif t.get('mark') == 'critical':  
 marks.append(pytest.mark.critical)  
 params.append(  
 pytest.param(  
 t,  
 id=t['name'],  
 marks=marks  
 )  
 )  
 metafunc.parametrize('test\_def', params)  
  
@pytest.fixture(scope='function')  
def db\_cursor(request):  
 host = request.config.getoption('--db-host')  
 port = request.config.getoption('--db-port')  
 db = request.config.getoption('--db-name')  
 user = request.config.getoption('--db-user')  
 pwd = request.config.getoption('--db-pass')  
  
 conn = psycopg2.connect(  
 host=host,  
 port=port,  
 database=db,  
 user=user,  
 password=pwd  
 )  
 cur = conn.cursor()  
 yield cur  
 conn.close()  
  
def pytest\_sessionfinish(session, exitstatus):marker = session.config.getoption('-m')  
 suite = marker if marker in ('smoke', 'critical') else 'full'  
  
 raw\_dir = f'reports/allure\_raw\_{suite}'  
 out\_dir = f'reports/allure\_{suite}'  
 cmd = [  
 'allure'  
 'generate',  
 raw\_dir,  
 '--clean',  
 '--output', out\_dir,  
 '--single-file'  
 ]  
 subprocess.run(cmd, check=True)

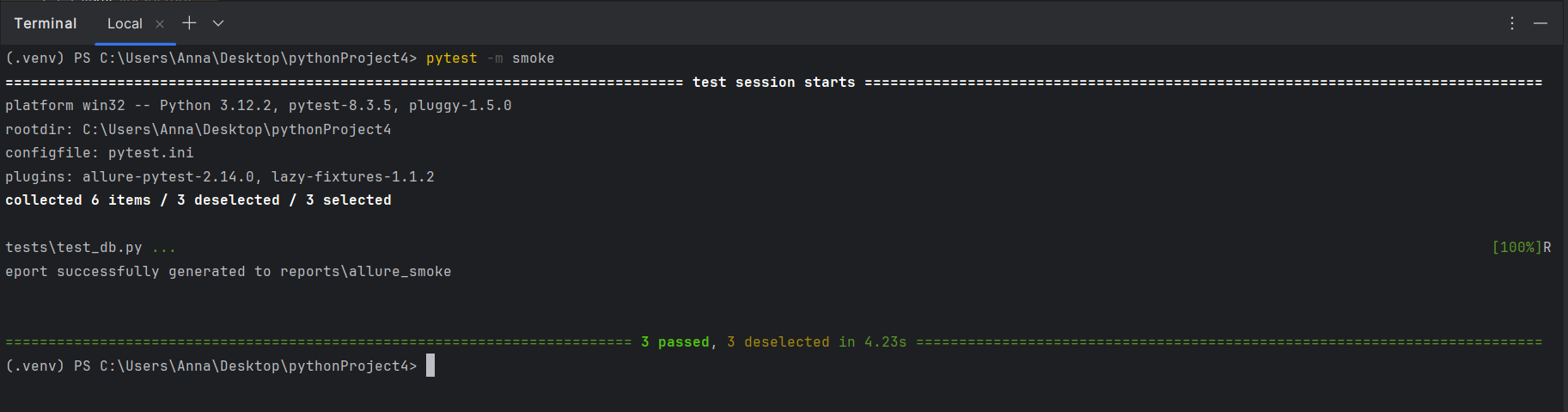
config\_SQL\_example.yaml:

tests:  
 - name: completeness\_check\_for\_users\_table  
 sql: >  
 SELECT COUNT(\*) FROM INFORMATION\_SCHEMA.TABLES  
 WHERE TABLE\_SCHEMA = 'public' AND TABLE\_NAME = 'users'  
 expected: 1  
 mark: smoke  
  
 - name: row\_count\_users  
 sql: "SELECT COUNT(\*) FROM users"  
 expected: 10  
 mark: critical  
  
 - name: null\_email\_check  
 sql: "SELECT COUNT(\*) FROM users WHERE email IS NULL"  
 expected: 0  
 mark: critical  
  
 - name: completeness\_check\_orders\_table  
 sql: >  
 SELECT COUNT(\*) FROM INFORMATION\_SCHEMA.TABLES  
 WHERE TABLE\_SCHEMA = 'public' AND TABLE\_NAME = 'orders'  
 expected: 1  
 mark: smoke  
  
 - name: row\_count\_orders  
 sql: "SELECT COUNT(\*) FROM orders"  
 expected: 100  
 mark: critical  
  
 - name: orphan\_orders\_check  
 sql: >  
 SELECT COUNT(\*) FROM orders o  
 LEFT JOIN users u ON o.user\_id = u.id  
 WHERE u.id IS NULL  
 expected: 0  
 mark: smoke

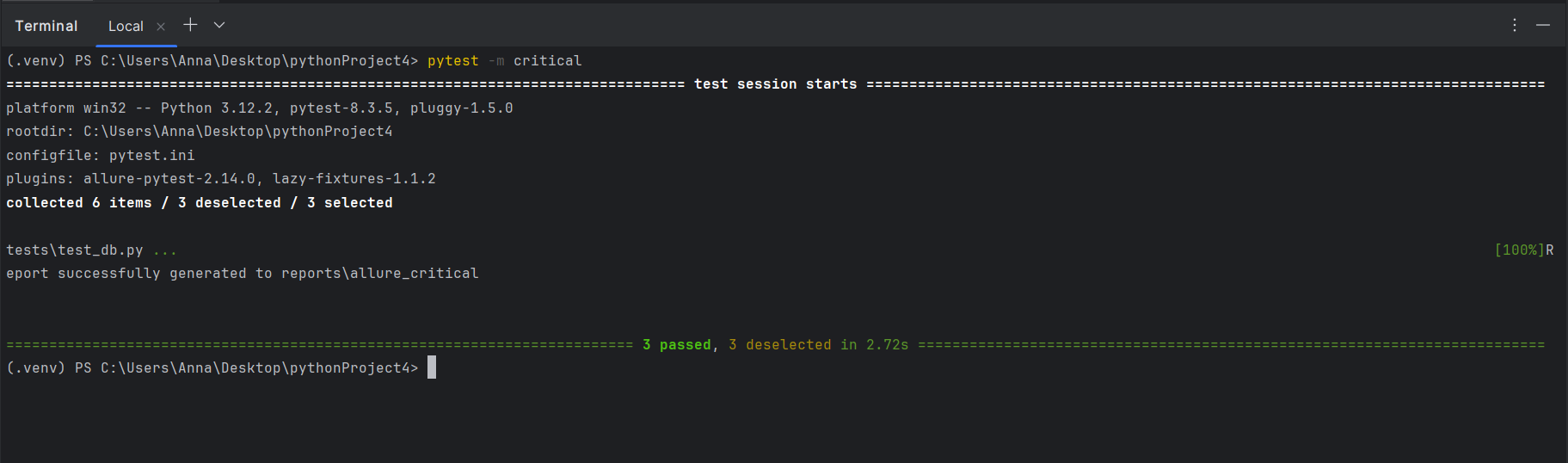
tests/tes\_db.py:

import pytest  
import allure  
  
  
def test\_sql\_queries(db\_cursor, test\_def, pytestconfig):  
 name = test\_def['name']  
 sql = test\_def['sql']  
 expected = test\_def['expected']  
 mark = test\_def.get('mark')  
 marker\_filter = pytestconfig.getoption('-m').strip()  
 if marker\_filter and mark and mark not in marker\_filter.split():  
 pytest.skip(f"Skipping {name}, not in -m {mark}")  
  
 with allure.step(f"Executing: {name}"):  
 db\_cursor.execute(sql)  
 result = db\_cursor.fetchone()[0]  
 assert result == expected, f"{name}: got {result}, expected {expected}"

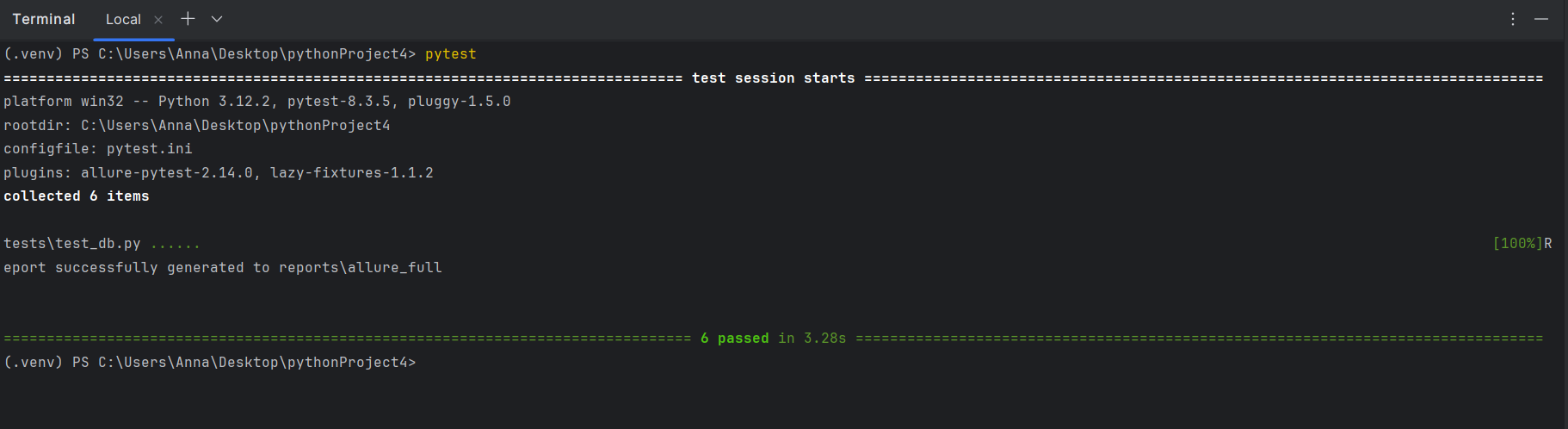
First run:

****

**Second run:**

****

**Third run:**

****