EDITH COWAN UNIVERSITY

CSI3105:
Software Testing
Module 3a: Test Automation and Junit Introduction

Creative
thinkers
made here.

Lecture Summary

The learning goal for this week are:

- To understand what is test automation and issues associated with test automation
- To use pytest for unit testing in Python

Lecture Summary

Video 3b

- Introduce some new terms and test cases

Video 3c

- Define the difference between manual and automated testing

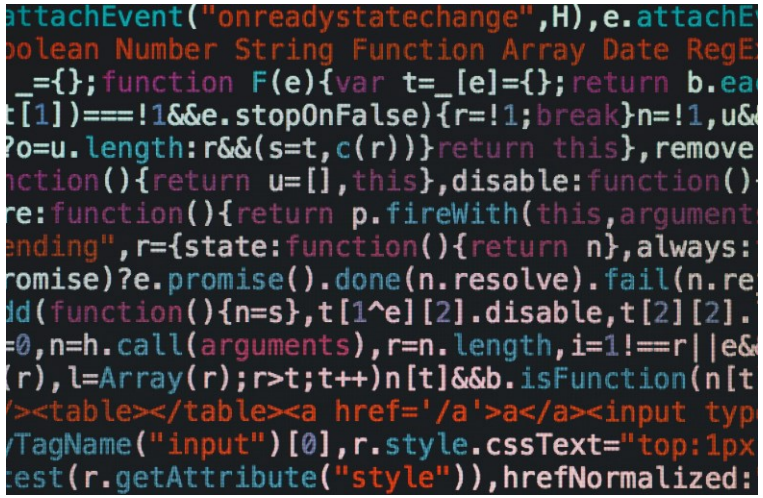
Video 3d

- Introduce pytest

Video 3e

- A deeper look at automated testing

Software Testing



- We code and introduce errors
- Potentially create a whole mess

…

- Software testing can be hard and laborious
- Some pieces of software can be harder than others to test

What is Software Testability?

EDITH COWAN UNIVERSITY

- The degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met (from Ammann & Offut, 2016)
- Testability of the software module is high, implies finding faults in the system via testing is easier
- Testability – how likely the testing will find a fault
- Some metrics associated with software testability:
  - Observability
  - Controllability
  - Availability
  - Simplicity
  - Stability

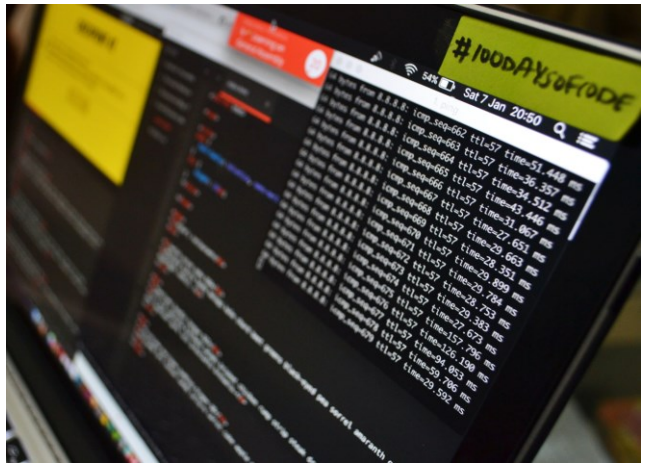
Observability

Observabili

How easy it is to observe the behavior of a program in terms of its outputs, effects on the environment and other hardware and software components

- Software that affects hardware devices, databases, or remote files have low observability

Hard!
Photo by Louis Reed on Unsplash



Easier!
Photo by Lewis Ngugi on Unsplash

Controllability

Controllabil

How easy it is to provide a program with the needed inputs, in terms of values, operations, and behaviors

- Easy to control software with inputs from keyboards
- Inputs from hardware sensors or distributed software is harder

Data abstraction reduces controllability and observability

Test Cases Introduction

ECU
EDITH COWAN UNIVERSITY

Controllability

- Giving input to the software under test

Observability

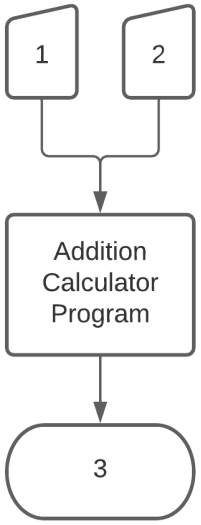
- Seeing what the software does after we give it the input

In a simple sense, to test software we must provide

- Some form of inputs to the system under test
- Observer and compare the output against what we expect to happen

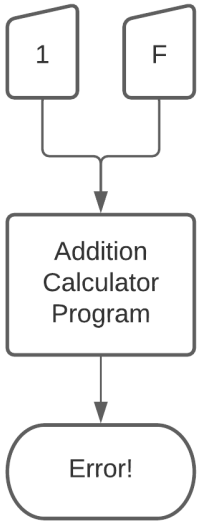
Test Cases

Test case #1:
Input: 1, 2
Expected output: 3



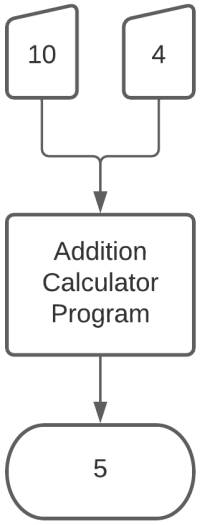
Expected output: 3
Observed output: 3
Passed

Test case #2:
Input: 1, F
Expected output: Error!



Expected output: Error!
Observed output: Error!
Passed

Test case #3
Input: 10, 4
Expected output: 14



Expected output: 14
Observed output: 5
Failed

Components of a Test Case

A test case is a multipart artifact with a definite structure

Input Values

- Test case values

The input values needed to complete an execution of the software under test

- Expected results

The result that will be produced by the test if the software behaves as expected

  - A *test oracle* uses expected results to decide whether a test passed or failed

Affecting Controllability and Observability

EDITH COWAN UNIVERSITY

Prefix values

Inputs necessary to put the software into the appropriate state to receive the test case values

Postfix values

Any inputs that need to be sent to the software after the test case values are sent

1. *Verification Values* : Values needed to see the results of the test case values
2. *Exit Values* : Values or commands needed to terminate the program or otherwise return it to a stable state

Putting Tests Together

EDITH COWAN UNIVERSITY

Test case	The test case values, prefix values, postfix values, and expected results necessary for a complete execution and evaluation of the software under test
Test set	A set of test cases
Executable test sc	A test case that is prepared in a form to be executed automatically on the test software and produce a report

Manual Testing

- Software tester essentially takes up the role of the end user
- Manual Testing

  Write test cases

  Set up the test environment/ Load Test data

  Execute tests

  Compare results

  Log tests results

  Clear up test environment

  Summarise results into a report

  Analysis of the test reports

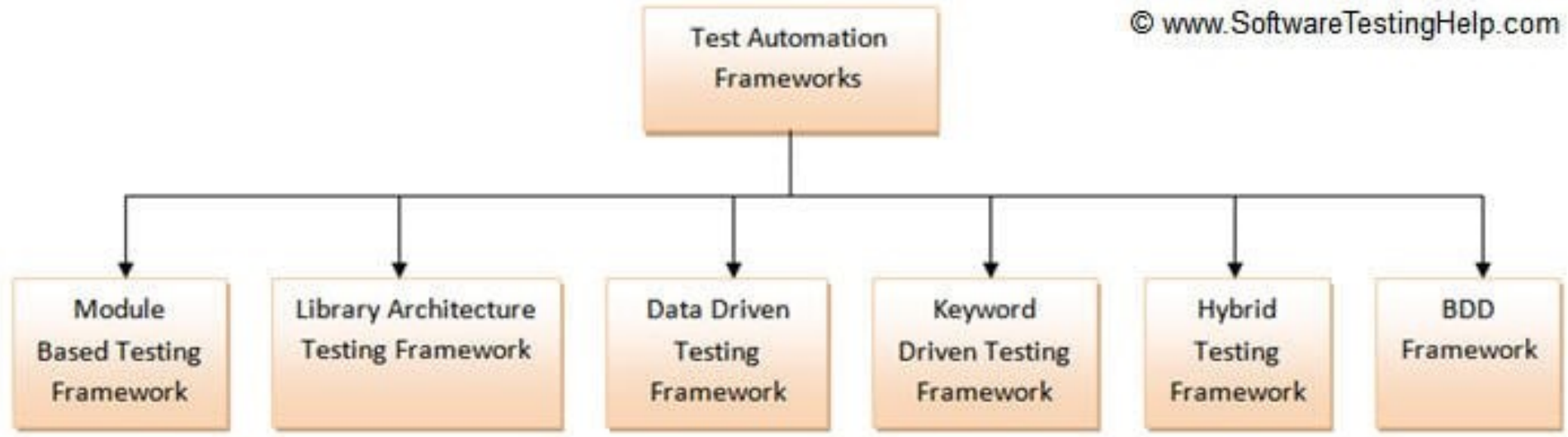
Automated Testing

EDITH COWAN UNIVERSITY

- Manual testing is good..
  - Can be a long and fatiguing process
  - Human resource intensive
  - This can introduce human error
- Test cases are executed with the assistance of tools, scripts, and software.
- Using software or tools we can execute many test cases without human intervention
  - Simulate thousands of concurrent users

Test Automation Framework

- An integrated system with a combination of sets the rules, protocols, standards and guidelines to be followed. Components (e.g. function libraries, test data sources, object details and various reusable modules) within this system are the small building blocks that need to be assembled to represent a testing process.
- It provides the basis of test automation and simplifies the automation effort.
- It is always application independent. It should be scalable and maintainable.



Test Automation Frameworks

Advantages of Test Automation frameworks

EDITH COWAN UNIVERSITY

- Modularity
- Understand-ability
- Maximum test coverage
- Code reuse
- Recovery test scenarios
- Low-cost maintenance
- Minimal manual intervention
- Ease of Reporting

Automated Testing

Definition: The act of conducting specific tests via automation (such as executing a set of regression tests) as opposed to conducting them manually.

Lets have a look at how we could automate some tests

What is pytest?

EDITH COWAN UNIVERSITY

- Open source Python testing framework used to write and run repeatable automated tests pytest is open source (pytest.org)
- Provides a structure for writing test drivers
- pytest features include:
  - Assertions for testing expected results (with Python's built-in assert keyword)
  - Fixtures for sharing common test data and setting up resources before tests run
  - Test discovery for automatically finding and running tests in your project
  - Plugins and extensions for additional functionality (e.g., parallel execution, coverage reports)
- pytest is widely used in industry
- pytest can be used as standalone Python scripts (from the command line) or integrated into IDEs such as PyCharm and VS Code

pytest Tests

EDITH COWAN UNIVERSITY

- pytest can be used to test …
  - … an entire Python object (such as a class instance)
  - … part of an object – a single function, method, or interacting methods
  - … interaction between several objects or components

- pytest is primarily intended for unit and integration testing, not system testing.
- Each test is written as a simple function using Python's def keyword, not requiring classes.
- A test file can contain one or more test functions.
- Test files include:
- A collection of test functions (each starting with test_)
- Fixtures to set up and tear down state before and after each test, or once for all tests using
  - @pytest.fixture, setup_module, or teardown_module.
- Get started at pytest.org

pytest

EDITH COWAN UNIVERSITY

- pytest Core:
  - Handles test discovery, execution, and reporting, forming the backbone of the framework.
- pytest Fixtures:
  - Provides a flexible mechanism for setting up and tearing down resources needed by tests, including sharing test data and states.
- pytest Plugins:
  - Supports extensions that add functionality such as parallel execution, code coverage, and more, making pytest highly customizable and powerful.

pytest

EDITH COWAN UNIVERSITY

- Why use pytest?
  - Can’t we just use the debugging tool in our IDE?
  - Can’t we add print statements in our code?

- These methods are good for finding logic errors
  - Can work for extremely small-scale projects
  - Limited to our own judgment
  - If we’ve already made the error, will we be able to catch it?

- pytest provides automated testing
  - Can test many scenarios in one execution

Writing Tests for pytest

EDITH COWAN UNIVERSITY

- Each test method uses assert statements to check conditions and report to pytest whether the test passed or failed.
- The results from these assertions are used by pytest to provide detailed feedback to the user, either in the command line interface or through an IDE like PyCharm or VS Code.
- All assertions are built into Python and do not need a separate import when using pytest.
- A few representative assertion examples in pytest include:
  - assert 1 == 1 (Checks if a condition is true)
  - assert 2 + 2 == 5, "This test failed because 2+2 is not 5" (Provides a custom failure message)
  - assert False, "This test is forced to fail"
- For more info about pytest assertions, check out: https://docs.pytest.org

Test Fixtures

EDITH COWAN UNIVERSITY

- A test fixture in pytest sets up the data needed for every test
  - Objects and variables that are used by more than one test
  - Initialisations using fixtures (like prefix values)
  - Tear-down or reset operations after tests (like postfix values)

- Different tests can use the fixture without sharing state, ensuring each test runs in isolation.

- In pytest:
  - Fixtures are created using the @pytest.fixture decorator.
  - Objects can be initialized in a fixture function and passed to tests.
  - Fixtures can also handle cleanup using request.addfinalizer or by using yield statements for teardown after each test.

Basic Template for Writing Test

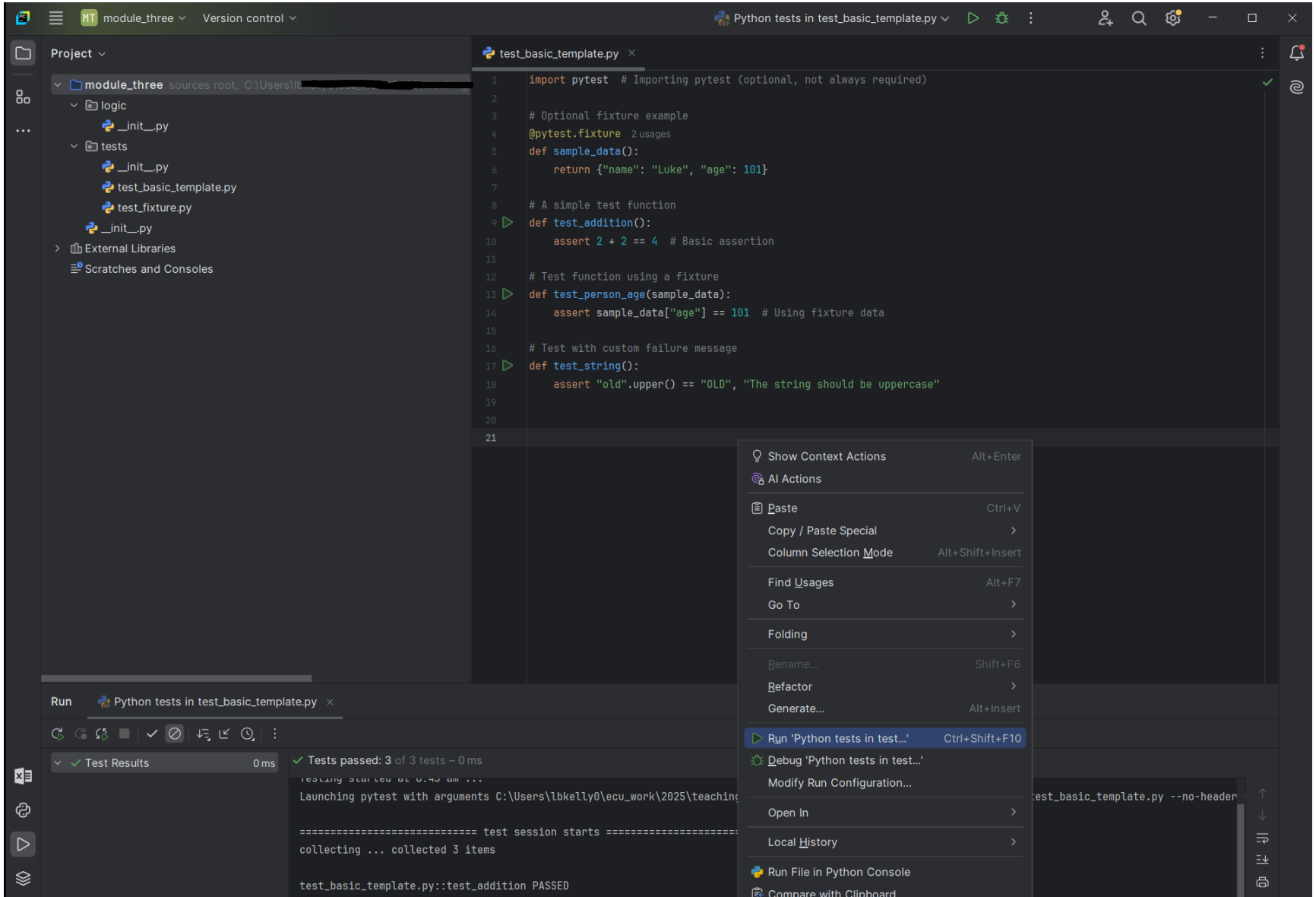
```python
import pytest  # Importing pytest (optional, not always required)

# Optional fixture example
@pytest.fixture  2 usages
def sample_data():
    return {"name": "Luke", "age": 101}

# A simple test function
def test_addition():
    assert 2 + 2 == 4  # Basic assertion

# Test function using a fixture
def test_person_age(sample_data):
    assert sample_data["age"] == 101  # Using fixture data

# Test with custom failure message
def test_string():
    assert "old".upper() == "OLD", "The string should be uppercase"
```

```
✓ Tests passed: 3 of 3 tests – 0 ms
Launching pytest with arguments C:\User[redacted]

============================= test session starts ==============================
collecting ... collected 3 items

test_basic_template.py::test_addition PASSED                             [ 33%]
test_basic_template.py::test_person_age PASSED                           [ 66%]
test_basic_template.py::test_string PASSED                               [100%]

============================== 3 passed in 0.02s ===============================

Process finished with exit code 0
```

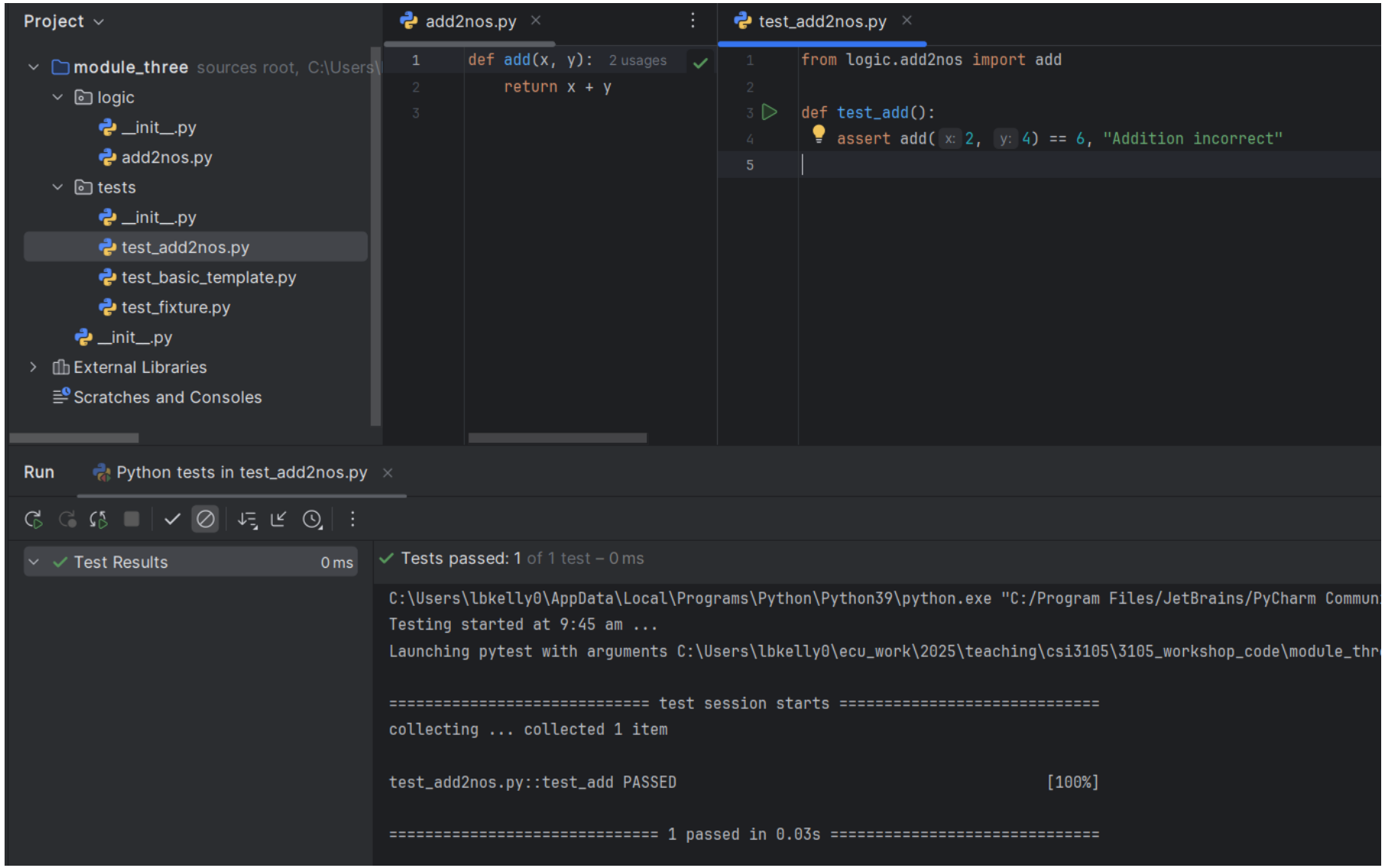
Key Elements of a pytest Test:

- Test functions start with test_ so pytest can automatically discover and run them.
- Assertions use Python's built-in assert statement to check expected outcomes.
- Fixtures (optional) are used to provide reusable test data or setup code.
- No need for boilerplate code like test classes (unless necessary), making pytest simpler and more Pythonic.

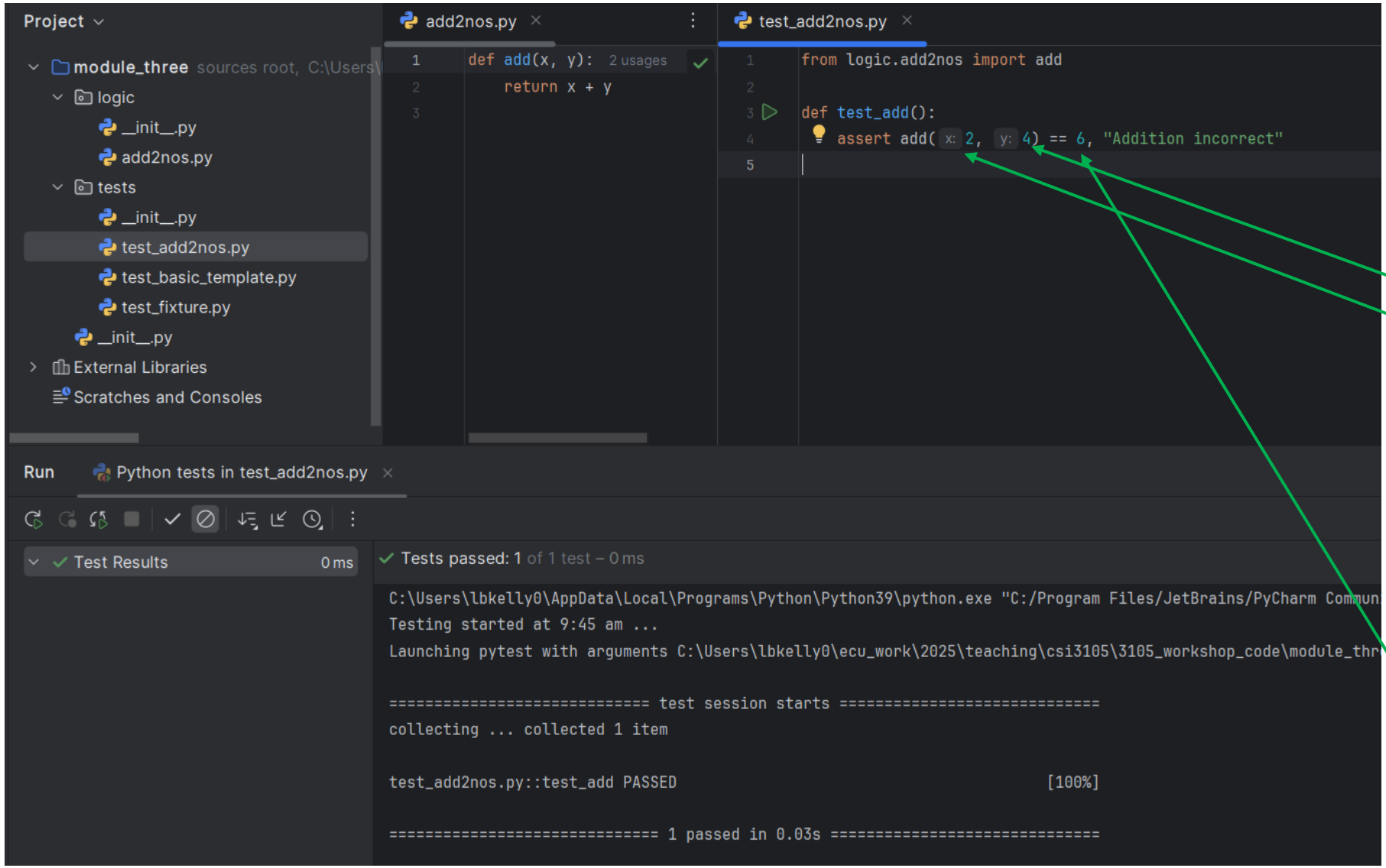
Simple Example



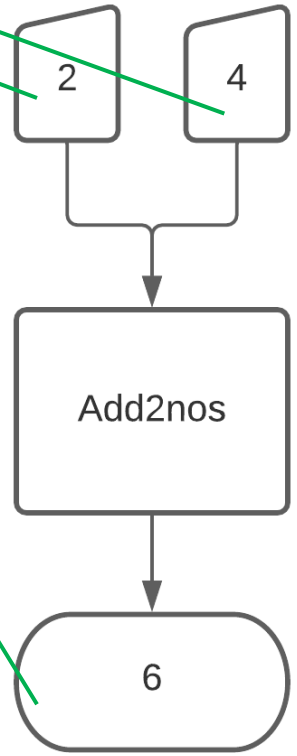
Simple Example



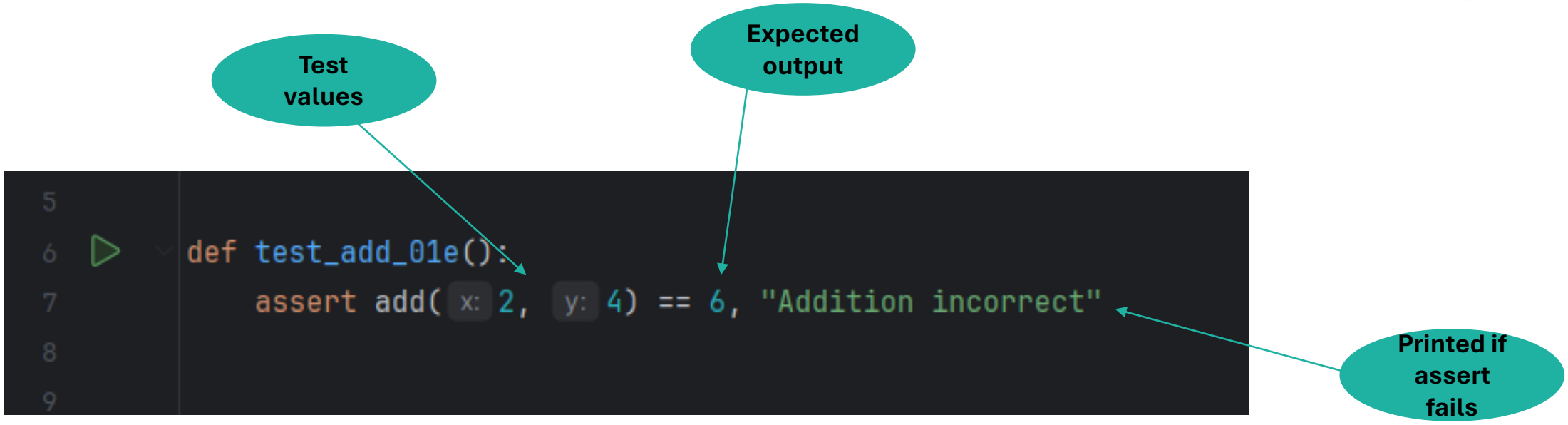
Simple Example



Test case #1:
Input: 2, 4
Expected output: 6

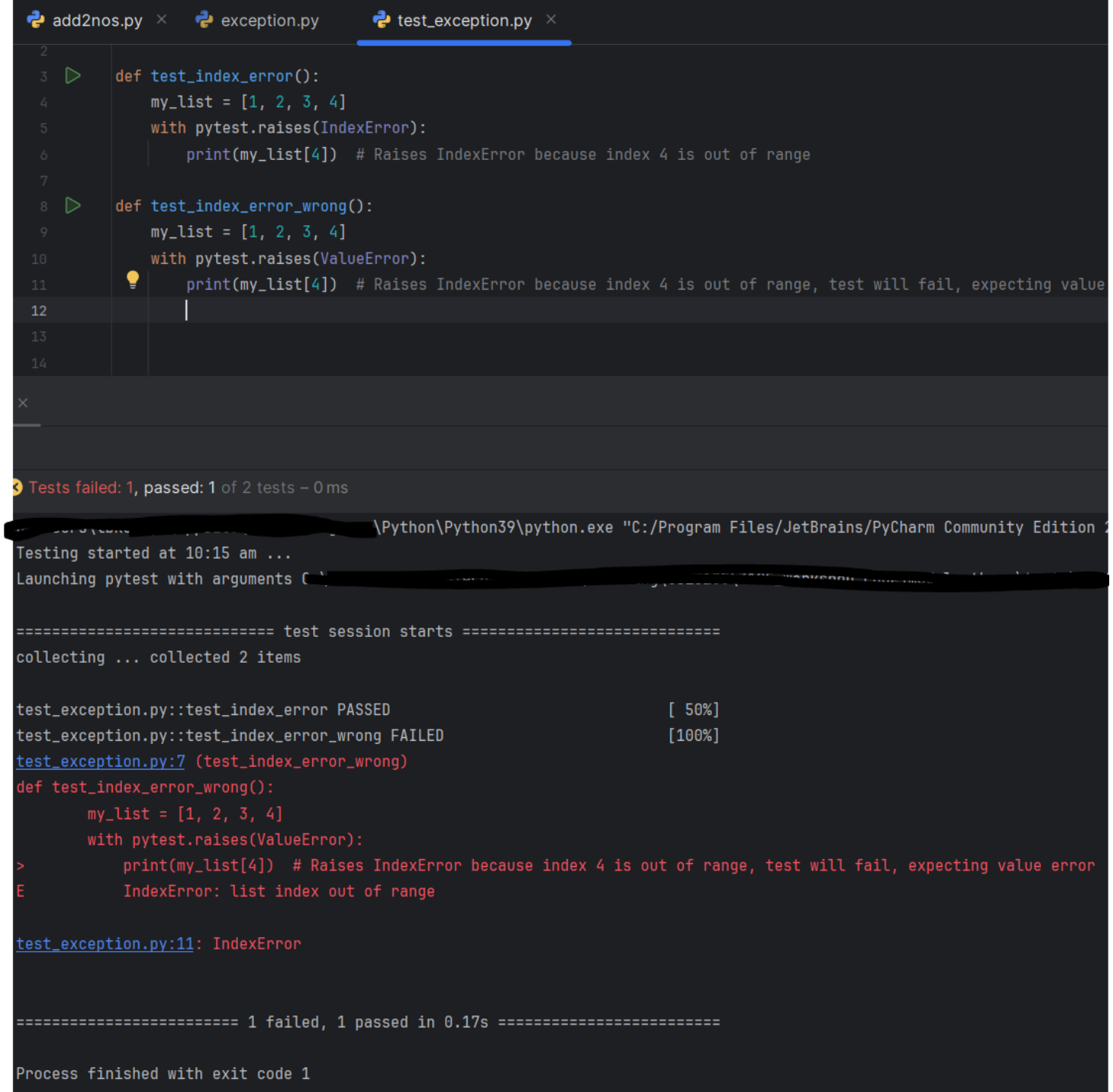


Simple Example



Test id	Test Objective	Technique	Expected Input	Expected Output
test_add_01e	Valid input returns the correct value	E	2,4	6

Exception Testing



EDITH COWAN UNIVERSITY

Parameterised Tests

test_params.py

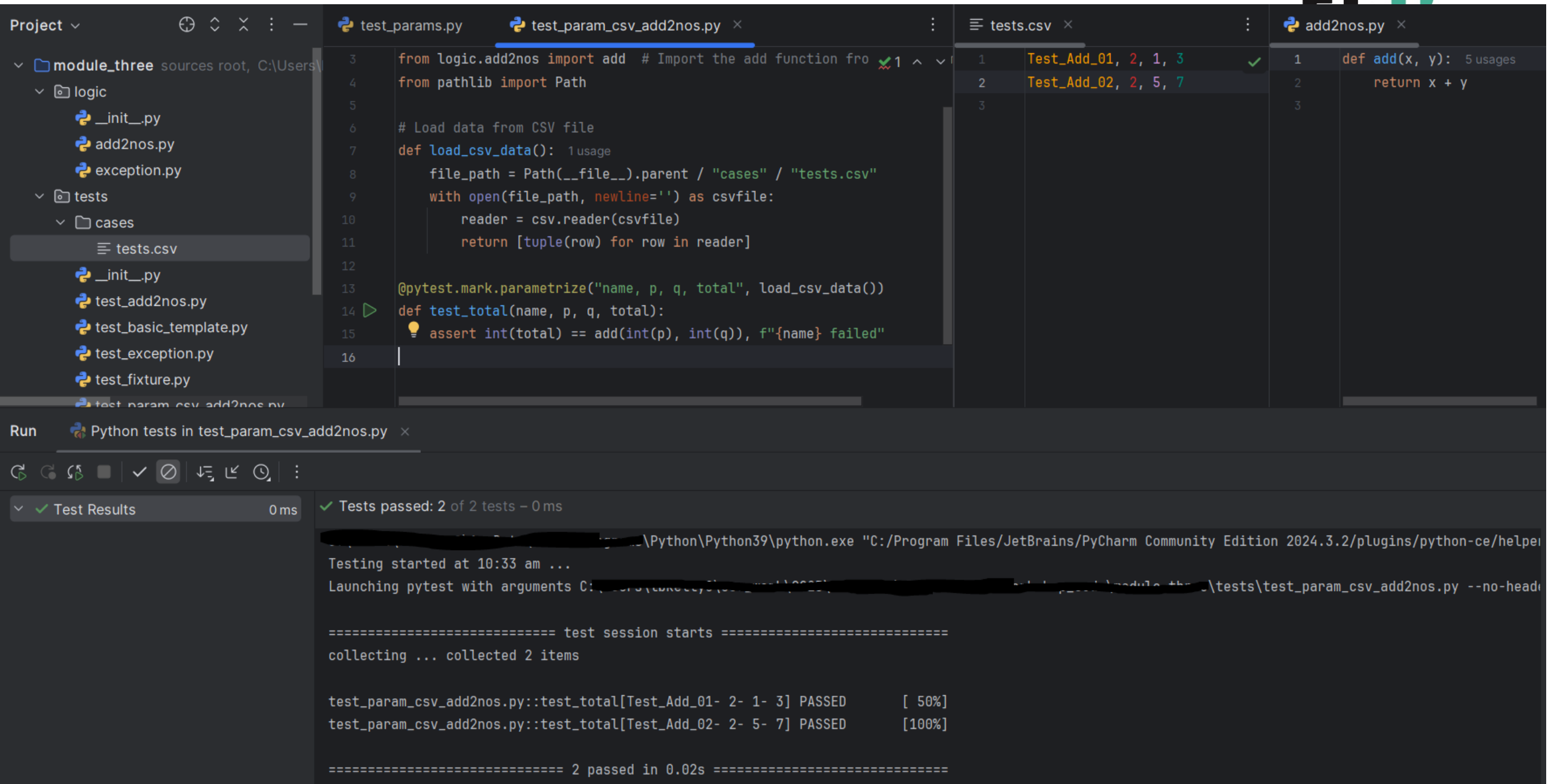
```python
import pytest

@pytest.mark.parametrize("candidate", ["racecar", "radar", "able was I ere I saw elba", "Luke"])
def test_is_palindrome(candidate):
    assert candidate == candidate[::-1], f"{candidate} is not a palindrome"

```

Tests failed: 1, passed: 3 of 4 tests – 0 ms

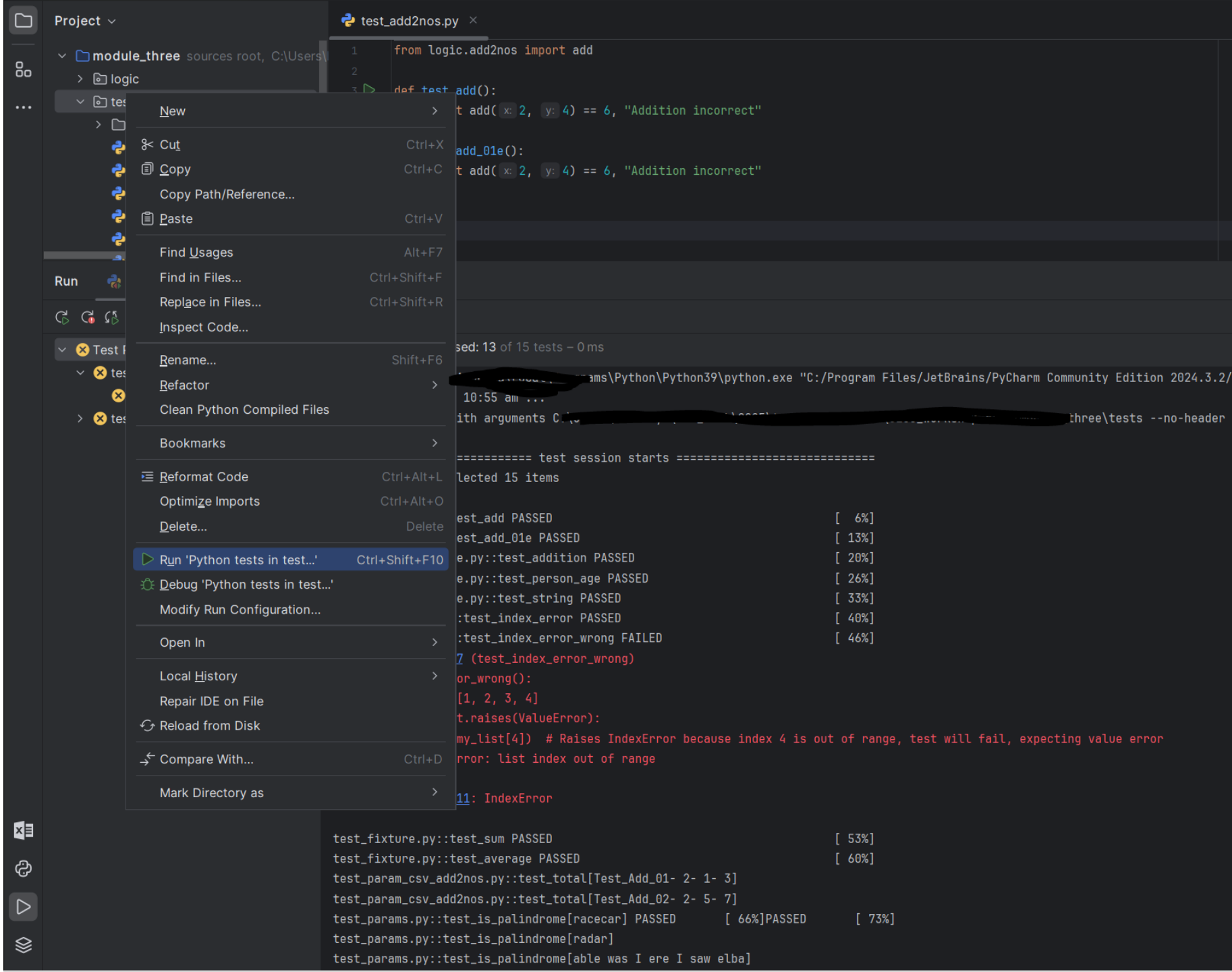
```
C:\[redacted]\Python\Python39\python.exe "C:/Program Files/JetBrains/PyCharm Community Edition 2024.3.2/pl
Testing started at 10:21 am ...
Launching pytest with arguments [redacted]params.py

============================= test session starts ==============================
collecting ... collected 4 items

test_params.py::test_is_palindrome[racecar]
test_params.py::test_is_palindrome[radar]
test_params.py::test_is_palindrome[able was I ere I saw elba]
test_params.py::test_is_palindrome[Luke]

========================= 1 failed, 3 passed in 0.10s ==========================
PASSED                    [ 25%]PASSED                    [ 50%]PASSED     [ 75%]FAILED                    [100%]
test_params.py:2 (test_is_palindrome[Luke])
'Luke' != 'ekuL'

Expected :'ekuL'
Actual   :'Luke'
<Click to see difference>

candidate = 'Luke'
```

EDITH COWAN UNIVERSITY

Parameterized Tests from CSV



Executing a Test Suite



Ending notes on automated testing

From the Ropota Andrei / Automated Testing slides

Don‘t worry too much about these slides

Refer to these when you are starting your report for the group assignment

Automated testing advantages

Costs and efficiency

- Detection of the errors that reached production phase (with regression tests)
- Multiple users simulation
- Reusable of the old scripts -> creation of the new scripts is reduce
- automatic execution of performance tests in the beginning of the production ->less costs for improving the performance

Time economy

- quick analysis in case of changing of environment parameters
- short duration of the testing cycles
- better estimation for test planning
- a large number of tests can be executed over night
- quick generation of testing preconditions

Quality increase

- automatic compare of results
- more consistent results due to repeating tests

Limitation of Automated Testing

Most of the times an Automated Testing system can't tell if something "looks good" on the screen or when a pictogram or a window is not displayed well

There are a bunch of problems that can appear when trying to automate the testing process:

- Unrealistic expectations (e.g. expectation that automated tests will find a lot of errors)
- Poor testing experience
- Maintenance of automated tests

Automated testing will never replace definitely the manual testing

Tests that should not be automated are:

- tests that are executed very rare
- where the system is very unstable
- tests that can be verified easily manually but hardly automated
- tests that need physical interaction

Automated testing vs. Manual testing

Pros of **Automated testing**

- If a set of tests must be ran repeatedly, automation is a huge win
- It offers the possibility to run automation against code that frequently change to catch regressions
- Offers the possibility to add a large test matrix (e.g. different languages on different OS platforms)
- Automated tests can be run the same time on different machines, whereas manual tests must be run sequentially
- It offers more time for the test engineer to invoke greater depth and breadth of testing, focus on problem analysis, and verify proper performance of software following modifications and fixes
- Combined with the opportunity to perform programming tasks, this flexibility promotes test engineer retention and improves his morale

- Cons of **Automated testing**
  - Costs - Writing the test cases and writing or configuring the automate framework that is used costs more initially than running the test manually.
  - Some tests can't be automated – manual tests are needed

Automated testing vs. Manual testing (2)

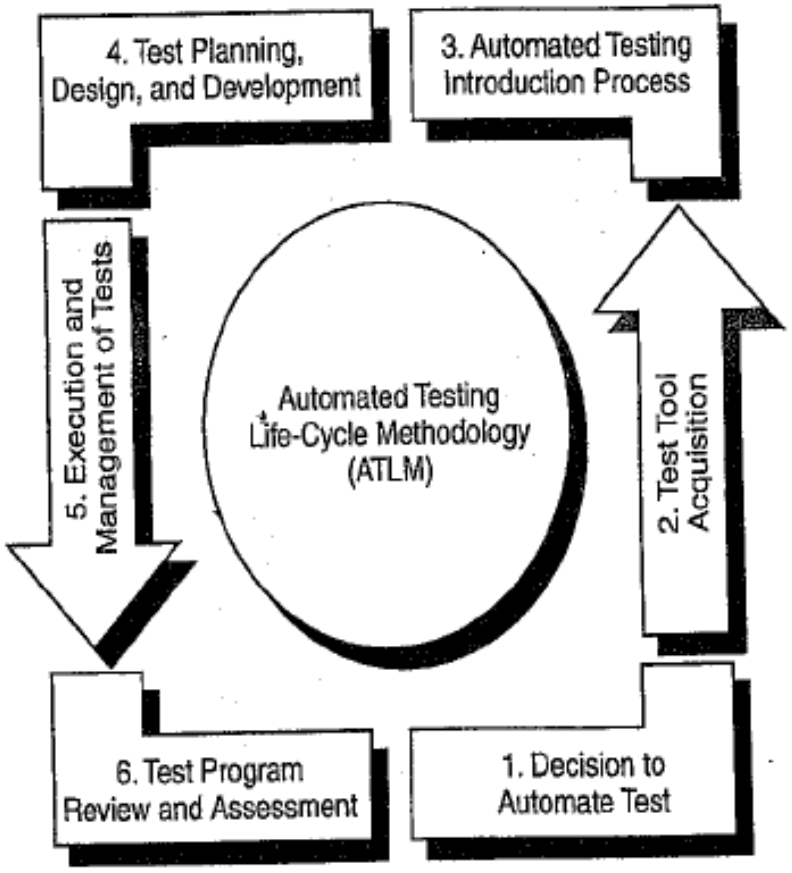
Pros of **Manual testing**

- If the test case runs once or twice most likely is a manual test. Less cost than automating it.
- It allows the tester to perform more ad-hoc (random tests). Experience has proven that more bugs are found via ad-hoc (Experience Based Testing) than via automation testing. And more time the testers spends playing with the feature, the greater are the chances of finding real user problems.

- Cons of **Manual testing**
  - Running tests manually can be very time consuming
  - The manual tests are requiring more people and hardware
  - Each time there is a new build, the tester must rerun all required tests - which after a while would become very boring and tiresome.

Automated Test Life-Cycle Methodology (ATLM)

1. Decision to Automate Test
2. Test Tool Acquisition
3. Automated Testing Introduction Process
4. Test Planning, Design and Development
5. Execution and Management of Tests
6. Test Program Review and Assessment



Decision to Automate Test: Overcoming False Expectations

EXPECTATION

Automatic test plan generation

Test tool fits all

Imminent test effort reduction

Tool ease of use

Universal application of test automation

100% test coverage

REALITY

- No tool can create automatically a comprehensive test plan
- No single tool can support all operating systems environments and programming languages
- Initial use of an automated test tool can actually increase the test effort
- Using an automated tool requires new skills, additional training is required
- Not all the tests required for a project can be automated (e.g. some test are physically impossible; time or cost limitation)
- It is impossible to perform an exhaustive testing of all the possible inputs (simple or combination) to a system

Decision to Automate Test: Benefits of Automated Testing

EDITH COWAN UNIVERSITY

Production of a reliable system	Improved requirements definition Improved performance/load/stress testing Improved partnership with development team Improved system development life cycle
Improvement of the quality of the test effort	Improved build verification testing (smoke testing) Improved regression testing Improved multiplatform/software compatibility testing Improved execution of the repetitive tests Improved focus on advanced test issues Execution of tests that manual testing can't accomplish Ability to reproduce software defects After-hours testing
Reduction of test effort and minimization of schedule	

Test Tool Acquisition

- Identify which of the various tool types suit the organization system environment, considering:
  - the group/department that will use the tool;
  - the budget allocated for the tool acquisition;
  - the most/least important functions of the tool etc.
- Choose the tool type according to the stage of the software testing life cycle
- Evaluate different tools from the selected tool category
- Hands-on tool estimation – request product demonstration (evaluation copy)
- Following the conclusion of the evaluation process, an evaluation report should be prepared

Business analysis phase	business modeling tools configuration management tools defect tracking tools
Requirements definition phase	requirements management tools requirements verifiers tools
Analysis and design phase	database design tools application design tools
Programming phase	syntax checkers/debuggers memory leak and run-time error detection tools source code or unit testing tools static and dynamic analyzers
Testing phase	test management tools network testing tools GUI testing tools (capture/playback) non-GUI test drivers load/performance testing tools environment testing tools

Automated Testing Introduction Process – Test Process Analysis

Process review	Test process characteristics (goals, strategies, methodologies) have been defined and they are compatible with automated testing Schedule and budget allows process implementation The test team is involved from the beginning of SDLC
Test goals	Increase the probability that application under test will behave correctly under all circumstances Increase the probability that application meets all the defined requirements Execute a complete test of the application within a short time frame
Test objectives	Ensure that the system complies with defined client and server response times Ensure that the most critical end-user paths through the system perform correctly Incorporate the use of automated test tools whenever feasible Perform test activities that support both defect prevention and defect detection Incorporate the use of automated test design and development standards to create reusable and maintanable scripts
Test strategies	Defect prevention (early test involvement, use of process standards, inspection and walkthroughs) Defect detection (use of automated test tools, unit/integration/system/acceptance test phase)

Test Planning

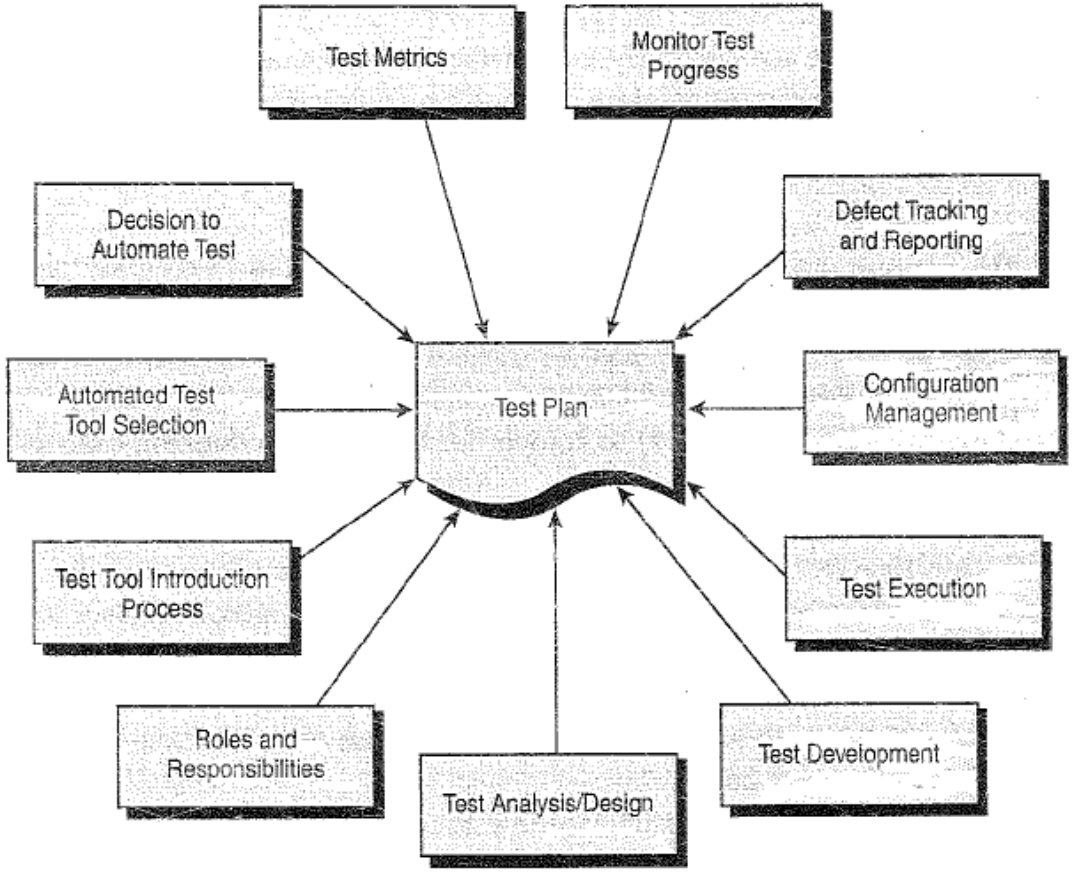
- The test planning element of the ATLM incorporates the review of all activities required in the test program
- It ensures that testing processes, methodologies, techniques, people, tools, schedule and equipment are organized and applied in an efficient way
- Key elements: planning associated with project milestone events, test program activities and test program-related documentation.
- The following must be accomplished:

- the technical approach for these elements is developed;

- personnel are assigned

- performance timelines are specified in the test program schedule.

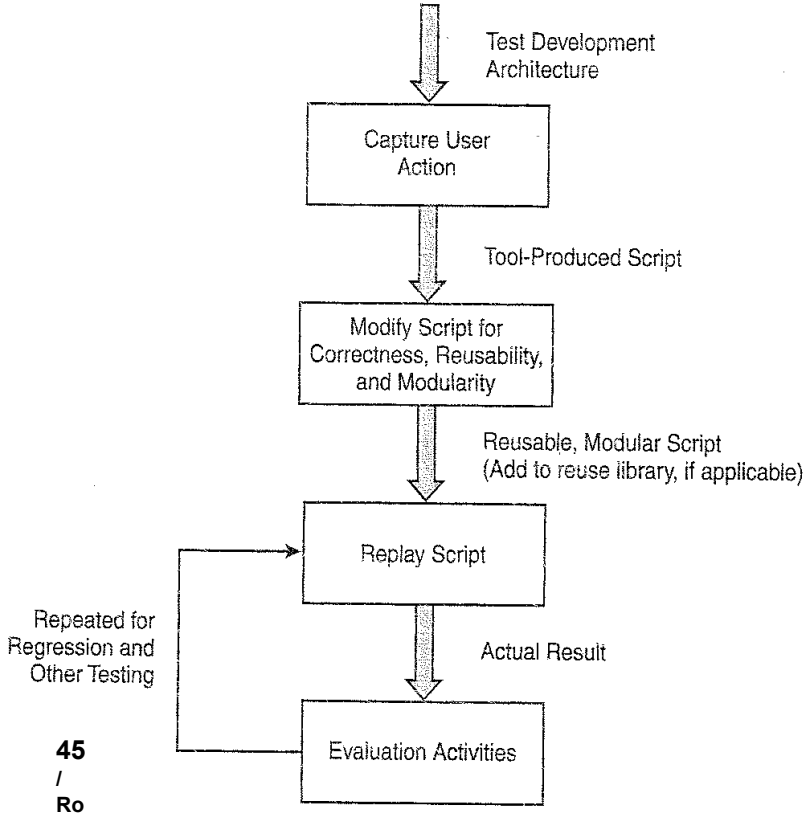
- Test planning is not a single event, but rather a process. It is the document that guides test execution through to a conclusion, and it needs to be updated frequently to reflect any changes.



Test Design

Black-Box Test Techniques	Automated Test Tools
Random Testing	GUI Test Tools
Regression Testing	GUI/Server Test Tools
Stress/Performance Testing	Load Test Tools
Security Testing	Security Test Tools
Data Integrity Testing	Data Analysis Tools
Configuration Testing	Multiplatform Test Tools
Functional Testing	Load/GUI/Server Test Tools
User Acceptance Testing	GUI Test Tools
Usability Testing	Usability Measurement Tools
Alpha/Beta Testing	Load/GUI/Server Test Tools
Boundary Value Analysis	Develop program code to perform tests
Backup and Recoverability Testing	Load/GUI/Server Test Tools

Test Development



Development Guideline Topics	Description
Design-to-Development Transition	Specify how design and setup activities will be translated into test development action
Reusable Test Procedures	Test procedures need to be reusable for highest test program return on investment
Data	Avoid hard-coding data values into scripts
Capture/Playback	Outlines on how to apply the use of capture/playback recording
Maintainable Test Procedures	A test procedure whose defects are easy to remove and can easily be adapted to meet new requirements
Test Script Documentation	Test script documentation is important for test procedure maintainability
Naming Standards	Defines the standard naming convention for test procedures
Modularity	Guidelines for creating modular test scripts
Global Files	Globally declared functions are available to any procedure
Constants	Use of constants in order to support maintainable procedure

Test Execution