Mindmap to Learn pytest

* **Introduction to pytest**
  + **Overview**: pytest is a mature full-featured Python testing tool that helps write simple and scalable test cases for functional and unit testing.
  + **Installation**: Install pytest using pip: pip install pytest.
* **Basic Concepts**
  + **Test Functions**: Write test functions using the test\_ prefix.
    - Example: def test\_example(): assert 1 + 1 == 2.
  + **Running Tests**: Run tests using the pytest command.
    - Example: pytest to run all tests, pytest test\_file.py to run tests in a specific file.
* **Assertions**
  + **Basic Assertions**: Use Python's assert statement to verify test outcomes.
    - Example: assert x == y.
  + **Advanced Assertions**: Use pytest's assertion introspection for detailed error messages.
    - Example: assert a == b, f"Expected {a} but got {b}".
* **Fixtures**
  + **Defining Fixtures**: Use the @pytest.fixture decorator to create fixtures.
    - Example: @pytest.fixture def setup\_data(): return {"key": "value"}.
  + **Using Fixtures**: Pass fixtures as parameters to test functions.
    - Example: def test\_data(setup\_data): assert setup\_data["key"] == "value".
  + **Fixture Scope**: Define the scope of fixtures (function, class, module, session).
    - Example: @pytest.fixture(scope="module") def setup\_module(): pass.
* **Parametrization**
  + **Parametrize Tests**: Use @pytest.mark.parametrize to run a test with multiple sets of data.
    - Example: @pytest.mark.parametrize("input,expected", [(1, 2), (3, 4)]) def test\_add(input, expected): assert input + 1 == expected.
* **Markers**
  + **Custom Markers**: Create and use custom markers for organizing tests.
    - Example: @pytest.mark.slow def test\_slow\_function(): pass.
  + **Using Markers**: Select tests to run using markers.
    - Example: pytest -m slow to run tests marked as slow.
* **Plugins**
  + **Using Plugins**: Extend pytest's functionality with plugins.
    - Example: pytest-cov for coverage reporting, pytest-xdist for parallel test execution.
  + **Writing Plugins**: Create custom plugins for pytest.
    - Example: Define hooks in conftest.py.
* **Test Discovery**
  + **Naming Conventions**: Follow pytest's naming conventions for test discovery.
    - Example: Test files should start with test\_, and test functions should start with test\_.
  + **Configuration**: Configure test discovery using pytest.ini, tox.ini, or setup.cfg.
    - Example: [pytest] testpaths = tests.
* **Test Coverage**
  + **Coverage Reporting**: Measure code coverage using pytest-cov.
    - Example: pytest --cov=your\_module.
  + **Coverage Configuration**: Configure coverage reporting.
    - Example: Use .coveragerc to specify coverage settings.
* **Mocking**
  + **Mock Objects**: Use unittest.mock or pytest-mock to mock objects and functions.
    - Example: from unittest.mock import Mock; mock = Mock().
  + **Patching**: Temporarily replace objects for the duration of the test.
    - Example: from unittest.mock import patch; @patch('module.function') def test\_func(mock\_func): pass.
* **Assertions for Exceptions**
  + **Exception Assertions**: Use pytest.raises to assert that exceptions are raised.
    - Example: with pytest.raises(ValueError): func\_that\_raises().
* **Parameterizing Fixtures**
  + **Parameterized Fixtures**: Use @pytest.fixture(params=...) to create parameterized fixtures.
    - Example: @pytest.fixture(params=[1, 2, 3]) def numbers(request): return request.param.
* **Test Order**
  + **Controlling Test Order**: Use pytest-ordering to control the order of test execution.
    - Example: @pytest.mark.run(order=1) def test\_first(): pass.
* **Customizing Test Runs**
  + **Command-Line Options**: Add custom command-line options.
    - Example: def pytest\_addoption(parser): parser.addoption("--custom", action="store", default="value").
  + **Hooks**: Use pytest hooks to customize behavior.
    - Example: def pytest\_configure(config): pass.
* **Debugging Tests**
  + **Debugging**: Use pytest with pdb for debugging.
    - Example: pytest --pdb.
  + **Verbose Output**: Get detailed output using the -v option.
    - Example: pytest -v.
* **Test Parameterization with Fixtures**
  + **Combining Parametrize and Fixtures**: Use both for more complex test setups.
    - Example: @pytest.mark.parametrize("input,expected", [(1, 2), (3, 4)]) def test\_func(input, expected, setup\_fixture): pass.
* **Temporary File Handling**
  + **Temp Directory**: Use tmp\_path fixture for creating temporary files and directories.
    - Example: def test\_temp\_file(tmp\_path): file = tmp\_path / "test.txt"; file.write\_text("content").
* **Test Selection**
  + **Selecting Tests**: Run specific tests using node IDs.
    - Example: pytest tests/test\_module.py::test\_function.
* **Parallel Testing**
  + **Running Tests in Parallel**: Use pytest-xdist for parallel test execution.
    - Example: pytest -n auto.
* **Best Practices**
  + **Code Organization**: Organize tests in a separate directory (e.g., tests).
    - Example: tests/test\_module.py.
  + **Modular Tests**: Write modular and independent tests.
    - Example: Avoid dependencies between tests.
  + **Consistent Naming**: Use consistent naming for tests and fixtures.
    - Example: test\_function for test functions, setup\_fixture for fixtures.