Why IDE(INTEGRATED DEVELOPMENT ENVIRONMENT)

- **IDE: application used to develop software.**
  - GUI for writing code
  - Syntax highlighting - making some words bold or italic
  - Code completion
  - Code, compile, debug and run your programs
  - IDE knows these programming language rules/syntax, so Flag any syntax, rules issue in our code
  - Integrate with Git(Version Control System)

Pytest Framework

- **Why Framework**:
  - Structure our test
  - Added helpful features
  - More focus on writing and adding tests

- **About Pytest**:
  - Test framework based on Python language
  - Easy to write, execute and generate test reports
  - Different types and levels of testing
  - Used by developers and QA team

Install Pkgs & Test Naming

- **Installing Python Packages:**
  - Install the latest version of Pytest-
    - >pip install pytest
  - Can specify version while installing:
    - >pip install pytest==5.4.3
  - See all pkgs in your python installation:
    - >pip3 list /or pip list
  - Put all pkgs list in a file:
    - >pip freeze >requirements.txt

- **Test Naming Conventions:**
  - Name all the test modules(test files) with test_<somename> or or <somename>_test.py.
  - Name all the test functions with test_<somename>
  - Test classes should be named Test<somename>.
  - Properly group all the tests into test classes and packages(keep __init__.py inside the folder)

Running Tests and Outputs

- **Running tests:**
  - pytest <folder-name>
  - pytest <file-name>
  - Note: Enable virtual env first if running from cmd prompt.
  - Pytest options:
    - Option: -v, verbose mode, increases the verbosity level. Prints a message each time a module is initialized, showing the place (filename or built-in module) from which it is loaded.

- **Understanding Test Outputs :**
  - --lf, --last-failed : only re-run the failures.
  - --ff, --failed-first : to run the failures first and then the rest of the tests.
  - Ref doc pytest_cache: https://docs.pytest.org/en/latest/cache.html

Assertion & Test Discovery

- **Assertions:**
  - Way of validating the tests
  - Use operators like ==, !=, <=, >= for asserting.
  - You can use the above operators for validating strings and numbers.
  - assert 1, means assert True
  - assert 0, means assert False. This is going to fail your test.
  - In Python, multiplication, division has higher precedence than add, subtract.
  - assert with 'in' operator to validate if a value is within a tuple, list or string.
  - Good Test writing Practice: Try to add only 1 assert in single test.

- **Test Discovery:**
  - Given no arguments, pytest looks at your current directory and all subdirectories for test files and runs the test code it finds.
  - If you give pytest a filename, a directory name, or a list of those, it looks there instead of the current directory.
  - Each directory listed on the command line is recursively traversed to look for test code.

Pytest Markers

- **Markers:**
  - Use markers for grouping our tests.
  - A test can have more than one marker, and a marker can be on multiple tests.
  - E.g. marking a test:
    - @pytest.mark.sanity, @pytest.mark.uitest, etc
  - While running tests, use pytest -m sanity, to run only test marked with "sanity"
  - Markers running supports: and, or, not, parentheses operators.
    - E.g. pytest -m "sanity and uitest"
  - Can define markers in module/file level:
    - pytestmark=[pytest.mark.smoke, pytest.mark.sanity]
  - For stopping the warnings, need add/define the markers in pytest.ini in root folder

Pytest Cmd Line options

- **Cmd line options:**
  - See all options with pytest --help.
  - Few of them below:
    - -m MARKEXPR only run tests matching given mark expression.
    - -x, --exitfirst, exit instantly on first error or failed test.
    - --tb=no, no traceback
    - --maxfail=num exit after first num failures or errors.
    - -q, --quiet decrease verbosity.
    - --collect-only only collect tests, don't execute them.
    - --version display pytest lib version and import information.

Parameterize & Fixtures

- **Parametrized Tests:**
  - Parameterized tests allow a developer to run the same test over and over again on different values.
  - Parameterized testing is when we want to do data driven testing. For example, we want to test the login page or process, with multiple input values for e.g. say 1000 different username and password conditions.
  - This lets you increase the coverage without having to add much more test code.
    - E.g. @pytest.mark.parametrize("test_input,expected", [("5+5", 10), ("5-5", 0), ("7*8", 56)])

- **Fixtures:**
  - Fixtures is used to get data ready for our tests. Fixtures are functions that are run by pytest before (and sometimes after) the actual test functions.
  - We can use fixtures to get a data set for the tests to work on. For e.g. you want to setup DB connection, or initialize webdriver and test browser if talking in terms of selenium UI tests.
  - We can put fixtures in individual test files if we want the fixture to only be used by tests in that file.
  - Proper way of use is to have fixtures in conftest.py to be used by multiple test files.
  - Fixture functions can have a any name.
  - Call fixture by using in test-function argument. e.g. def test_someTest(fixture_name)
  - Mark fixture using @pytest.fixture()
  - Its not mandatory but we can return something from fixture using return statement.

Parameterize & Fixtures

- **Fixtures:**
  - Also another way of calling fixture:
    - @pytest.mark.usefixtures("fixture_name")
    - In this case, return from fixture cannot be used. So only use it to execute some code within fixture.

- **Setup/Tear down and using multiple fixture:**
  - "yield": By using a yield statement instead of "return", all the code after the yield statement serves as the teardown code and are run.
  - We can have code to shutdown connections, etc after the "yield" stmt.
  - We can use/pass multiple fixture in same test function.

- **Sharing Fixtures:**
  - Use conftest.py - put the fixtures to share across multiple test files.
  - We can have more conftest.py files in subdirectories of the top tests directory. If we do, fixtures defined in these lower-level conftest.py files will be available to tests in that directory and subdirectories.
  - Don’t import conftest from anywhere. The conftest.py file gets read by pytest, and is considered a local plugin.
  - Function- pytest_configure, vars within this are available within all test functions in that module.