Pytest Notes:

Here are detailed notes with examples for the listed concepts related to running Pytest and using its features effectively:

**How to Run Pytest**

1. **Basic Execution**:
   * To run all tests in the current directory and its subdirectories:

pytest

**How to Run Pytest Using Verbose (-v)**

* **Verbose Output**:
  + Use -v for more detailed output, showing each test function as it is executed:

pytest -v

**-s - Detailed Print Statements in Console**

* **Printing Statements**:
  + Use -s to display detailed print statements from your test functions directly in the console:

pytest -s

**How to Run Selective Pytests**

* **Running Selective Tests**:
  + Pytest allows running specific tests in multiple ways.

**Substring of the Test Name (-k)**

* **Run Tests by Substring**:
  + Execute tests whose names match a substring:

arduino

pytest -k "substring"

* + Example: Running tests containing "login" in their name:

arduino

pytest -k "login"

**Marker (-m)**

* **Run Tests Marked with a Marker**:
  + Mark tests using @pytest.mark decorator and run them by marker:

python

import pytest

@pytest.mark.login

def test\_login():

# Test login functionality

pass

* + Run tests marked with @pytest.mark.login:

pytest -m login

**Mark Few Cases with Same Name to Other Test**

* **Grouping Tests**:
  + Group tests using markers and run them together:

python

import pytest

@pytest.mark.login

def test\_login():

# Test login functionality

pass

@pytest.mark.login

def test\_logout():

# Test logout functionality

pass

* + Run all tests marked with @pytest.mark.login:

pytest -m login

**How to Run Pytest Present in a File**

* **Run Tests from a Specific File**:
  + Specify the path to a specific test file to run its tests:

bash

pytest path/to/test\_file.py

**How to Run Pytests Present Under a Folder**

* **Run Tests from a Folder**:
  + Specify the path to a directory containing tests to run all tests within that directory and its subdirectories:

bash

pytest path/to/folder

**Fixtures**

* **Using Fixtures**:
  + Fixtures provide a way to set up preconditions and clean up after tests.
  + Example:

python

import pytest

@pytest.fixture

def setup\_database():

# Setup database

yield

# Teardown database

* + Use the fixture in a test function:

python

def test\_database\_operations(setup\_database):

# Test database operations

pass

**Yield**

* **Yield in Fixtures**:
  + yield in a fixture marks the point where the test function is paused and resumed after the test function completes.
  + Example:

python

@pytest.fixture

def setup\_teardown():

# Setup

yield

# Teardown

**Conftest**

* **Using Conftest**:
  + conftest.py files define fixtures, hooks, and configurations for test sessions and test functions.
  + Place conftest.py in a directory to apply fixtures and hooks to tests in that directory and subdirectories.

**Parallel Execution**

* **Running Tests in Parallel**:
  + Pytest supports running tests in parallel for faster execution.
  + Configuration and plugins (like pytest-xdist) are used for parallel execution.

**Reports - HTML**

* **Generating HTML Reports**:
  + Generate HTML reports to visualize test results:

css

pytest --html=report.html

**How to Stop Your Execution After n Failures**

* **Stop Execution After Failures**:
  + Stop test execution after a certain number of failures:

css

pytest --maxfail=n

**How to Skip a Test**

* **Skipping Tests**:
  + Skip a test using @pytest.mark.skip decorator or conditional statements:

python

import pytest

@pytest.mark.skip(reason="Test not implemented yet")

def test\_feature\_x():

pass

**How to Use Xfail Option**

* **Expected Failures (Xfail)**:
  + Mark tests as expected failures using @pytest.mark.xfail:

python

import pytest

@pytest.mark.xfail

def test\_function():

# Test that is expected to fail

pass