SOFTWARE TESTING ASSIGNMENT

Practical Exercises

1) **Unit Testing: Perform unit testing for a simple "Calculator" class that has methods for addition, subtraction, multiplication, and division. Write test cases for each method and ensure they pass successfully. Provide the code for both the test cases and the Calculator class.**

Unit Testing:

Unit testing is a software testing technique that focuses on evaluating individual units or components of a software application in isolation. A "unit" in this context typically refers to the smallest testable part of a software program, such as a function, method, or class. The main purpose of unit testing is to ensure that each unit of the code works as expected and that it performs its specific function correctly. It is done during the coding phase by the developers. To perform unit testing, a developer writes a piece of code (unit tests) to verify the code to be tested (unit) is correct.

Test Cases:

a) **Test Cases on addition:**
   [1] Verify that the addition function correctly adds two positive integers.
   [2] Verify that the addition function correctly adds two negative integers.
   [3] Verify that the addition function handles the addition of a positive and a negative integer correctly.
   [4] Verify that adding zero to a number returns the original number.
   [5] Verify that the addition function handles floating-point numbers correctly.
   [6] Test the addition of two large numbers to ensure that it doesn't result in overflow or precision issues.

b) **Test Cases on subtraction:**
   [1] Verify that the subtraction function correctly subtracts one positive integer.
   [2] Ensure that the subtraction function correctly handles the subtraction of one negative integer from another.
   [3] Verify that the subtraction function correctly handles the subtraction of a positive integer from a negative integer.
   [4] Verify that subtracting zero from a number returns the original number.
   [5] Verify that the subtraction function correctly handles floating-point numbers.

```
class TestCalculator(unittest.TestCase):
    def setUp(self):
        self.calculator = Calculator()

    def test_addition(self):
        self.assertEqual(self.calculator.add(2, 3), 5)  # 2 + 3 = 5
        self.assertEqual(self.calculator.add(-2, -3), -5) # (-2) + (-3) = -5
        self.assertEqual(self.calculator.add(-1, 1), 0)  # -1 + 1 = 0
        self.assertEqual(self.calculator.add(0, 0), 0)  # 0 + 0 = 0
        self.assertEqual(self.calculator.add(2, 0), 2) # 2 + 0 = 2
        self.assertEqual(self.calculator.add(2000000, 3000000), 5000000) # 1000000 + 2000000
= 5000000
        self.assertEqual(self.calculator.add(2.5, 3.5), 6.0) # 2.5 + 3.5 = 6.0

    def test_subtraction(self):
        self.assertEqual(self.calculator.subtract(5, 3), 2)  # 5 - 3 = 2
        self.assertEqual(self.calculator.subtract(-5, -3), -2) # (-5) - (-3) = -2
        self.assertEqual(self.calculator.subtract(-1, 1), -2)  # -1 - 1 = -2
        self.assertEqual(self.calculator.subtract(0, 0), 0)  # 0 - 0 = 0
        self.assertEqual(self.calculator.subtract(3.5, 2.5), 1.0) # 3.5 - 2.5 = 1.0

    def test_multiplication(self):
        self.assertEqual(self.calculator.multiply(2, 3), 6)  # 2 * 3 = 6
        self.assertEqual(self.calculator.multiply(-2, -4), 8) # -2 * -4 = 8
        self.assertEqual(self.calculator.multiply(8, -3), -24) # 8 * -3 = -24
        self.assertEqual(self.calculator.multiply(3.5, 2.5), 8.75)  # 3.5 * 2.5 = 8.75
        self.assertEqual(self.calculator.multiply(0, 5), 0)  # 0 * 5 = 0

    def test_division(self):
        self.assertEqual(self.calculator.divide(6, 2), 3)  # 6 / 2 = 3
        self.assertEqual(self.calculator.divide(-8, -2), 4) # -8 / -2 = 4
```

3. UI Lag During Search:

Description: While conducting a search for a specific artist or song, you notice a noticeable lag in the user interface (UI). When you start typing in the search bar, there is a delay before the search results appear.

Impact: Sluggish UI responsiveness can be frustrating for users, affecting the overall usability of the application.

4) **Regression Testing**: **Suppose you are working on a software project, and a new feature has been added to the application. Outline a regression testing strategy to ensure that the new feature does not introduce any new defects. Explain the types of test cases you would execute and any specific areas you would focus on.**

Regression Testing:

Regression testing is a type of software testing conducted after a code update to ensure that the update introduced no new bugs. This is because new code may bring in new logic that conflicts with the existing code, leading to defects. Usually, QA teams have a series of regression test cases for important features that they will re-execute each time these code changes occur to save time and maximize test efficiency.

Here's an outline of a regression testing strategy for this scenario:

1. **Identify Critical Use Cases:** Identify the critical use cases or key functionalities in the application that are most likely to be affected by the new feature. This can include core features and functionalities that many users rely on.
2. **Smoke Testing:** Start with a smoke test to ensure that the application still launches and the basic functionality works. This helps catch any critical issues early.
3. **Test Automation:** Automate relevant test cases, especially those that are part of your regression suite. Automated tests can be quickly executed and are less prone to human error.
4. **Execute a Full Regression Suite:** Run the entire regression test suite to verify that existing features are still working as expected. This suite should cover both the existing features and any new features or changes.
5. **Focus on Boundary and Edge Cases:** Pay special attention to boundary conditions and edge cases to ensure that the new feature doesn't create unexpected behavior in situations that are near the limits of the system's capabilities.
6. **Integration Testing:** Test the interaction between the new feature and existing features, making sure they work harmoniously together.
7. **Compatibility Testing:** Verify that the new feature works as expected across different browsers, operating systems, and devices, especially if the application has a web-based component.
8. **Performance Testing:** Conduct performance testing to ensure that the addition of the new feature doesn't lead to performance degradation or increased resource consumption.