Part 2: Unit testing activity

Test Case 1: Delete Character Function

To see if the function can handle errors, I first set the expected result incorrectly. Although the actual output was 123.4, the expected value was 123.45. The test failed as a result, which enabled me to validate the behaviour of the function. The correct result was then obtained by updating the expected value to 123.4. By testing the function's behaviour and resolving any problems that may occur, this method ensures that it operates as meant to.

```
// Test Case 1: Testing the delete character function for decimal number
describe('Testing the delete character function', function() {
    it('delete the last character of the display value', function() {
        let form = { display: { value: "123.45" } };
        calculator.deleteChar(form.display);
        // assert.equal(form.display.value, "123.45"); // Test fails
        assert.equal(form.display.value, "123.4"); // Test pass
    });
});
```

```
Testing the delete character function
  ✓ delete the last character of the display value


1 passing (4ms)
```

Test Case 2: Square Function

I purposely set the square function's expected result to 10 to make the test fail and confirm the function's behaviour. I changed the expected value to 25 after the failure to pass the test. This method adheres to best practices in test-driven development and guarantees the function's accuracy.

```
// Test Case 2: Testing the square function for 5^2
describe('Testing the square function', function() {
    it('should calculate the square of the input value correctly', function() {
        let form = { display: { value: "5" } };
        calculator.square(form);
        assert.equal(form.display.value, "25");
    });
});
```

```
Testing the square function
  ✓ should calculate the square of the input value correctly


2 passing (5ms)
```

Test Case 3: Square Root Function

With an expected output of 2, I used the input value 4 to test the square root function. To verify the function's behaviour, I first set the expected result to 0 to cause the test to fail. To make sure the function computes square roots correctly, I then changed the expected value to 2.

```
// Test Case 3: Testing the square root function
describe('Testing the square root function', function() {
    it('should calculate the square root correctly', function() {
        let form = { display: { value: "4" } };
        calculator.sqrt(form);
        assert.equal(form.display.value, "2");
    });
});
```

```
Testing the square root function
  ✓ should calculate the square root correctly


3 passing (7ms)
```

Test Case 4: Natural Logarithm Function (ln)

By setting the expected value to 1, I created a failure condition for the natural logarithm function (ln). As a result, the test failed, confirming the behavior of the function. To find out whether the absolute difference between the calculated and expected values, taking rounding errors into account, was within 0.0001, I used assert.isTrue.

```
describe('Testing the ln function', function() {
    it('should calculate ln 2 correctly', function() {
        let form = { display: { value: "2" } };
        calculator.ln(form);
        assert.isTrue(Math.abs(form.display.value - 0.6931) < 0.0001); // ln(2) ≈ 0.6931
    });
});
```

```
Testing the ln function
  ✓ should calculate ln 2 correctly


4 passing (10ms)
```

Test Case 5: Exponential Function (e)

The test failed because I entered an incorrect expected value of 1 rather than 2.71828. To make sure the output of the exp function is accurate, a failure condition was created. I

verified correctness using assert.isTrue with a margin of 0.0001 after changing the expected value to 2.71828.

```
describe('Testing the exponential function', function() {
    it('should calculate e^x correctly', function() {
        let form = { display: { value: "1" } };
        calculator.exp(form);
        assert.isTrue(Math.abs(form.display.value - 2.71828) < 0.0001); //  e (≈ 2.71828)
    });
});
```

```
Testing the exponential function
  ✓ should calculate e^x correctly


5 passing (11ms)
```

Test Case 6: +- Function

The input value's sign is changed by the sign change function. I changed a positive number to a negative and a negative number to a positive to test two scenarios. I tested with changing -3 to 3 and 2 to -2. To make sure the test failed, I first set the expected results to 2 and -3 incorrectly. I then changed what I expected to pass. Using both positive and negative inputs, this method confirms that the function is correct.

```
// Test Case 6: Testing the sign change function
describe('Testing the sign change function', function() {
    it('should change the sign of a positive number to negative', function() {
        let form = { display: { value: "2" } };
        calculator.changeSign(form.display);
        assert.equal(form.display.value, "-2");
    });
    it('should change the sign of a negative number to positive', function() {
        let form = { display: { value: "-3" } };
        calculator.changeSign(form.display);
        assert.equal(form.display.value, "3");
    });
});
```

```
Testing the sign change function
  ✓ should change the sign of a positive number to negative
  ✓ should change the sign of a negative number to positive


7 passing (13ms)
```

Test Case 7: Sine Function

The input value's sine in radians is calculated by the sine function. I used sin (0) and sin (pi/2) as my test cases. To make sure the test failed and validate the behavior of the function, I first used incorrect expected values. The test was then passed by applying the proper values. This ensures that, for standard inputs, the sine function operates accurately. I purposefully checked an incorrect condition for sin(pi/2) using assert.isFalse to make sure the test failed before confirming the correct result.

```
// // Test Case 7: Testing the sine function
describe('Testing the sine function', function() {
    it('should return 0 for sin(0)', function() {
        let form = { display: { value: "0" } };
        calculator.sin(form);
        assert.equal(form.display.value, "0");
    });
    it('should return approximately 1 for sin(π/2)', function() {
        let form = { display: { value: "1.5708" } }; // π/2 in radians
        calculator.sin(form);
        assert.isTrue(Math.abs(form.display.value - 1) < 0.0001);
    });
});
```

```
Testing the sine function
  ✓ should return 0 for sin(0)
  ✓ should return approximately 1 for sin(π/2)


9 passing (13ms)
```

Test Case 8: Cosine Function

The cosine of an input value in radians is calculated by the cosine function. I experimented with cos(0) and cos(pi). Creating a failure condition first, then fixing the anticipated outcomes, was my approach. I handled precision errors for cos(pi) by using assert.isTrue with a tolerance of 0.0001.

```
// Test Case 8: Testing the cosine function
describe('Testing the cosine function', function() {
    it('should return 1 for cos(0)', function() {
        let form = { display: { value: "0" } };
        calculator.cos(form);
        // assert.equal(form.display.value, "0"); // Test Fails
        assert.equal(form.display.value, "1"); // Test Pass
    });
    it('should return approximately -1 for cos(π)', function() {
        let form = { display: { value: "3.1416" } }; // π in radians
        calculator.cos(form);
        // assert.isTrue(Math.abs(form.display.value + 3.1416) < 0.0001); // Test Fails
        assert.isTrue(Math.abs(form.display.value + 1) < 0.0001); // Test Pass
    });
});
```

```
Testing the cosine function
  ✓ should return 1 for cos(0)
  ✓ should return approximately -1 for cos(π)


11 passing (15ms)
```

Test Case 9: Tangent Function

Two cases, tan (0) and tan(pi/4), were tested. By leaving the expected values erroneous, I first attempted to fail the tests before fixing them to pass. In order to ensure accurate results in both cases, I used assert.equal() to confirm the output for tan(0) and assert.isTrue() to see if the output for tan(pi/4) was close to 1.

```
// Test Case 9: Testing the tangent function
describe('Testing the tangent function', function() {
    it('should return 0 for tan(0)', function() {
        let form = { display: { value: "0" } };
        calculator.tan(form);
        // assert.equal(form.display.value, "1"); // Test Fails
        assert.equal(form.display.value, "0"); // Test Pass
    });
    it('should return approximately 1 for tan(π/4)', function() {
        let form = { display: { value: "0.7854" } }; // π/4 in radians
        calculator.tan(form);
        // assert.isTrue(Math.abs(form.display.value - 0.7854) < 0.0001); // Test Fails
        assert.isTrue(Math.abs(form.display.value - 1) < 0.0001); // Test Pass
    });
});
```

```
Testing the tangent function
  ✓ should return 0 for tan(0)
  ✓ should return approximately 1 for tan(π/4)


13 passing (12ms)
```

APPENDIX

```
const calculator = require('../script');
const assert = require('chai').assert;

// Test Case 1: Testing the modulas operator
describe('Testing the modulus operator %', function() {
    it('should calculate modulus correctly', function() {
        let form = { display: { value: "2%4" } };
        calculator.compute(form);
        // assert.equal(form.display.value, "0"); // Test fails
        assert.equal(form.display.value, "2"); // Test pass
    });
});

// Test Case 2: Testing the square function for 5^2
describe('Testing the square function', function() {
    it('should calculate the square of the input value correctly', function() {
        let form = { display: { value: "5" } };
        calculator.square(form);
        // assert.equal(form.display.value, "10"); // Test fails
        assert.equal(form.display.value, "25"); // Test pass
    });
});

// Test Case 3: Testing the square root function
describe('Testing the square root function', function() {
    it('should calculate the square root correctly', function() {
        let form = { display: { value: "4" } };
        calculator.sqrt(form);
        // assert.equal(form.display.value, "0"); // Test fails
```

```
        assert.equal(form.display.value, "2"); // Test pass
    });
});

// Test Case 4: Testing the ln function
describe('Testing the ln function', function() {
    it('should calculate ln 2 correctly', function() {
        let form = { display: { value: "2" } };
        calculator.ln(form);
        // assert.isTrue(Math.abs(form.display.value - 1) < 0.0001); // ln(2) ≈ 0.6931, Test fails
        assert.isTrue(Math.abs(form.display.value - 0.6931) < 0.0001); // ln(2) ≈ 0.6931, Test
pass
    });
});

// Test Case 5: Testing the e function
describe('Testing the exponential function', function() {
    it('should calculate e^x correctly', function() {
        let form = { display: { value: "1" } };
        calculator.exp(form);
        // assert.isTrue(Math.abs(form.display.value - 1) < 0.0001); //  e (≈ 2.71828), Test fails
        assert.isTrue(Math.abs(form.display.value - 2.71828) < 0.0001); //  e (≈ 2.71828), Test
pass
    });
});

// Test Case 6: Testing the sign change function
describe('Testing the sign change function', function() {
    it('should change the sign of a positive number to negative', function() {
        let form = { display: { value: "2" } };
        calculator.changeSign(form.display);
```

```
        // assert.equal(form.display.value, "2"); // Test fails
        assert.equal(form.display.value, "-2"); // Test Pass
    });
    it('should change the sign of a negative number to positive', function() {
        let form = { display: { value: "-3" } };
        calculator.changeSign(form.display);
        // assert.equal(form.display.value, "-3"); // Test fails
        assert.equal(form.display.value, "3"); // Test Pass
    });
});

// Test Case 7: Testing the sine function
describe('Testing the sine function', function() {
    it('should return 0 for sin(0)', function() {
        let form = { display: { value: "0" } };
        calculator.sin(form);
        // assert.equal(form.display.value, "1"); // Test fails
        assert.equal(form.display.value, "0"); // Test Pass
    });
    it('should return approximately 1 for sin(π/2)', function() {
        let form = { display: { value: "1.5708" } }; // π/2 in radians
        calculator.sin(form);
        // assert.isTrue(Math.abs(form.display.value - 1.5708) < 0.0001); // Test fails
        assert.isFalse(Math.abs(form.display.value - 1.5708) < 0.0001); // Test Pass
    });
});

// Test Case 8: Testing the cosine function
describe('Testing the cosine function', function() {
    it('should return 1 for cos(0)', function() {
        let form = { display: { value: "0" } };
```

```
        calculator.cos(form);
        // assert.equal(form.display.value, "0"); // Test Fails
        assert.equal(form.display.value, "1"); // Test Pass
    });
    it('should return approximately -1 for cos(π)', function() {
        let form = { display: { value: "3.1416" } }; // π in radians
        calculator.cos(form);
        // assert.isTrue(Math.abs(form.display.value + 3.1416) < 0.0001); // Test Fails
        assert.isTrue(Math.abs(form.display.value + 1) < 0.0001); // Test Pass
    });
});

// Test Case 9: Testing the tangent function
describe('Testing the tangent function', function() {
    it('should return 0 for tan(0)', function() {
        let form = { display: { value: "0" } };
        calculator.tan(form);
        // assert.equal(form.display.value, "1"); // Test Fails
        assert.equal(form.display.value, "0"); // Test Pass
    });
    it('should return approximately 1 for tan(π/4)', function() {
        let form = { display: { value: "0.7854" } }; // π/4 in radians
        calculator.tan(form);
        // assert.isTrue(Math.abs(form.display.value - 0.7854) < 0.0001); // Test Fails
        assert.isTrue(Math.abs(form.display.value - 1) < 0.0001); // Test Pass
    });
});
```