Question 1 (1 point) ✓ *Saved*

With shift-left testing, software is tested only at the end of the lifecycle by QA professionals.

- ○ True
- ◉ False

Question 2 (1 point) ✓ *Saved*

Which of the following best describes continuous delivery

- ○ A cloud platform designed to allow software to be easily sent to consumers
- ○ A strategy for testing applications using frameworks like Jest
- ○ A major release is packaged and sent at once
- ◉ Features are delivered incrementally in regular sprints

Question 3 (1 point) ✓ *Saved*

Which of the following best defines manual exploratory testing

- (●) A human tester will interact with the application like an end user to find bugs
- ( ) Testing how quickly your page loads and can handle server problems
- ( ) Writing automated scripts to easily test your application
- ( ) Testing the user interface of the application to ensure the visuals are high quality

Question 4 (1 point) ✓ *Saved*

What is an edge case test?

- (●) A test to check something that occurs very rarely
- ( ) Testing a specialized function of the application that must be tested to ensure the app works
- ( ) A basic testing time using a framework such as Jest to validate core features
- ( ) Testing a set of actions that the user performs on the application

Page 1:

1 ✓ 2 ✓ 3 ✓
4 ✓ 5 ✓ 6 ✓
7 ✓ 8 ✓ 9 ✓
10 ✓ 11 ✓ 12 ✓
13 ✓ 14 ✓ 15 ✓
16 ✓ 17 ✓ 18 ✓
19 ✓ 20 ✓ 21 ✓
22

Question 5 (1 point) ✓ *Saved*

Suppose we have a grade calculator application (an app that converts number grades to letters i.e. 95 to A+). If we were performing boundary value analysis, how would we test this application's grade input?

- ○ Test every single possibility automatically to ensure there are no errors
- ◉ Pick a number in the middle of the range to test (for example pick 82 in between A- and A)
- ○ Pick one random number at the boundary of the entire range from 1 to 100 to test
- ○ Test 50 random inputs as a sample test
- ○ Pick the exact numbers where the grades switch to a new letter to test

Question 6 (1 point) ✓ *Saved*

Suppose we are using a decision table to test whether a login input is working with 2 inputs, user name and password, where account name and password must both be correct. How many different test cases will we need to cover all possibilities.

- ○ 3
- ○ 2
- ○ 5
- ◉ 4

Question 7 (1 point) ✓ *Saved*

Entering an invalid login and verifying the system displays an error message instead of creating an account is a positive test case

- (●) True
- ( ) False

Question 8 (1 point) ✓ *Saved*

Which best defines regression testing?

- ( ) A test focusing on the smallest possible block of code that performs a function
- ( ) A form of manual exploratory testing that focuses on the UI
- ( ) Negative test cases such as verifying error messages are the main focus of the testing strategy
- (●) Test cases are run anytime code changes or a new feature is added

Question 9 (1 point) ✓ *Saved*

Which best defines a unit test?

- (●) It validates the smallest piece of code that can be separated from the rest of the application
- ( ) A test to verify whether external services such as a cloud database work together with your app

Question 10 (1 point) ✓ *Saved*

Which best defines an integration test?

- ○ It is a test strategy that focuses on manual exploration of the application including user flows
- ○ A test validating the smallest piece of code that can be separated from the rest of the application
- ○ It is a type of test to ensure that your servers will perform correctly under difficult situations
- ◉ A test to verify whether external services such as a cloud database work together with your app
- ○ It validates major features, ensuring that the application functions correctly as a whole

Question 11 (1 point) ✓ *Saved*

What is continuous integration?

- ○ A strategy for maintaining code where developers merge code together daily in a central repository
- ○ A testing method to verify that your application can be integrated with cloud services
- ○ A test of the entire end-to-end flow of the application
- ◉ A process where features are constantly being developed and pushed out to end

Question 12 (1 point) ✓ *Saved*

What is a smoke test?

- ○ A complete test of the entire end-to-end flow of the application
- ○ A test to verify whether external services such as a cloud database work together with your app
- ○ A test validating the smallest piece of code that can be separated from the rest of the application
- ◉ A minimal set of tests to verify the most important features are working correctly

Question 13 (1 point) ✓ *Saved*

Jenkins works as a continous integration (CI) server

- ◉ True
- ○ False

Question 14 (1 point) ✓ *Saved*

Continuous delivery and continuous deployment are two terms meaning the same concept

- ○ True
- ◉ False

Question 15 (1 point) ✓ *Saved*

Which Jest matcher would you use to verify two numbers equal each other?

- ○ toContain
- ○ None of the above
- ○ toMatch
- ◉ toBe

Question 16 (1 point) ✓ *Saved*

A Jest snapshot test is a method of performing integration testing

- ◉ True
- ○ False

Question 17 (1 point) ✓ *Saved*

Which example scenario would you likely test with Jest?

- ○ Verifying that your cloud service integrates with your main application
- ◉ Testing that the validation on the back end of the server performs correctly
- ○ Testing whether the state change correctly for a UI element
- ○ Testing the database to ensure data is secure

Question 18 (1 point) ✓ *Saved*

Which git command would you use to update a remote repository with local commits

- ○ git pull
- ○ git clone
- ○ git init
- ◉ git push

Question 19 (1 point) ✓ *Saved*

What does a Python unittest assert do?

- ○ Initializes the browser for the unittest
- ◉ Closes down the browser for the unittest
- ○ Installs selenium
- ○ Check if the output matches expectations

Question 20 (1 point) ✓ *Saved*

Which Selenium method can be used if you need to work with an element with the id: textbox

- ○ search_for_element(textbox)

Question 20 (1 point) ✓ *Saved*

Which Selenium method can be used if you need to work with an element with the id: textbox

- ○ search_for_element(textbox)
- ◉ find_element_by_id(textbox)
- ○ id(textbox)
- ○ selenium(textbox)

Midterm Short Answer

Question 21 (5 points) ✓ *Saved*

Suppose you are working on a contact form for a web application. The form has a div with the id "message" which will display a success message when the user creates an account with no errors.

Explain in your own words a step-by-step process using Selenium and a unit testing library, how you would write a unit test to verify that the success message functionality works correctly.

You do not have to write every line of code perfectly, the goal is to ensure you know the basic logic and steps required.

1. I have to setup an testing environment. It involves installing and importing the libraries

Question 21 (5 points) ✓ *Saved*

Suppose you are working on a contact form for a web application. The form has a div with the id "message" which will display a success message when the user creates an account with no errors.

Explain in your own words a step-by-step process using Selenium and a unit testing library, how you would write a unit test to verify that the success message functionality works correctly.

You do not have to write every line of code perfectly, the goal is to ensure you know the basic logic and steps required.

Question 22 (5 points) ✓ *Saved*

Suppose you are writing a tax application. It is designed to tell you what your tax rate will be based on your income. The different brackets are 0-24999, 25000 - 59,999 and 60,000+

Explain which testing framework you would use to easily test that entering a number in the application gives the correct tax bracket. In addition, explain how you would perform the test.

Suppose you are working on a contact form for a web application. The form has a div with the id "message" which will display a success message when the user creates an account with no errors.

Explain in your own words a step-by-step process using Selenium and a unit testing library, how you would write a unit test to verify that the success message functionality works correctly.

You do not have to write every line of code perfectly, the goal is to ensure you know the basic logic and steps required.

Sure, here is a step-by-step process for writing a unit test to verify the success message functionality of a contact form using Selenium and a unit testing library:

1. Set up a Selenium WebDriver instance to control a web browser. This can be done using the Selenium library for the programming language you are using.

2. Use the WebDriver instance to navigate to the page containing the contact form.

3. Fill out the contact form with valid input and submit it.

4. Wait for the page to load and verify that the success message is displayed in the "message" div using the WebDriver instance.

5. Assert that the text content of the "message" div matches the expected success message.

6. If the success message is not displayed or the text content does not match the expected message, fail the unit test with an appropriate error message.

Here's an example implementation in JavaScript using the Jest testing library and the Selenium WebDriver library:

```
const { Builder, By, Key, until } = require('selenium-webdriver');

const assert = require('assert');


describe('Contact form', function() {

 let driver;


 beforeEach(async function() {

  driver = await new Builder().forBrowser('chrome').build();

 });


 afterEach(async function() {
```

```
    await driver.quit();
  });

  it('displays success message when user creates account with no errors', async function() {
    // Navigate to the contact form page
    await driver.get('http://example.com/contact');

    // Fill out the form and submit it
    await driver.findElement(By.id('name')).sendKeys('John Doe');
    await driver.findElement(By.id('email')).sendKeys('johndoe@example.com');
    await driver.findElement(By.id('message')).sendKeys('This is a test message');
    await driver.findElement(By.id('submit')).click();

    // Wait for the success message to be displayed and verify its text content
    await driver.wait(until.elementLocated(By.id('message')), 5000);
    const messageElem = await driver.findElement(By.id('message'));
    const messageText = await messageElem.getText();
    const expectedMessageText = 'Your account was created successfully!';
    assert.strictEqual(messageText, expectedMessageText);
  });
});
```

Suppose you are writing a tax application. It is designed to tell you what your tax rate will be based on your income. The different brackets are 0-24999, 25000 - 59,999 and 60,000+ Explain which testing framework you would use to easily test that entering a number

in the application gives the correct tax bracket. In addition, explain how you would perform the test.

To easily test the tax bracket calculation functionality of the tax application, I would use a unit testing framework such as Jest or Mocha, which are popular testing frameworks for JavaScript applications.

Here's how I would perform the test using Jest as an example:

1. Write a test case that sets the income to a specific value and verifies that the correct tax bracket is returned. For example, the test case might look like this:

```
test('income of 30000 returns tax bracket of 25000-59999', () => {
  const income = 30000;
  const taxBracket = calculateTaxBracket(income);
  expect(taxBracket).toEqual('25000-59999');
});
```

1. Implement the calculateTaxBracket function that takes an income as input and returns the corresponding tax bracket based on the income. For example, the function might look like this:

```
function calculateTaxBracket(income) {
  if (income < 25000) {
    return '0-24999';
  } else if (income >= 25000 && income <= 59999) {
    return '25000-59999';
  } else {
    return '60000+';
  }
}
```

3. Run the test case and verify that it passes. If the test fails, investigate the issue and update the code as needed until the test passes.

4. Repeat steps 1-3 for different income values to ensure that the tax bracket calculation function works correctly across different income ranges.

Overall, using a unit testing framework like Jest makes it easy to write and run tests that verify the correct behavior of the tax bracket calculation function. By testing different income values and verifying that the correct tax bracket is returned, we can be confident that the function works as expected and the application will correctly calculate tax rates based on income.