**Q1**

## Role of Machine Learning in Software Test Automation

Machine learning (ML) has revolutionized the field of software test automation by enabling the development of more efficient, effective, and intelligent testing strategies. Here are some ways ML is being used in software test automation:

1. **Test Case Generation:** Machine Learning algorithms can generate test cases based on the software requirements, thereby reducing the need for manual test case creation improving test coverage.
2. **Test Case Prioritization:** ML can prioritize test cases based on their likelihood of finding defects, execution time, and criticality, ensuring that high-risk areas are thoroughly tested first, optimizing test execution time.
3. **Defect Prediction:** By analyzing historical defect data, ML algorithms can predict the likelihood of defects in specific areas of the software, allowing testers to focus their efforts on high-risk areas.
4. **Test Environment Management:** ML can be used to manage test environments, ensuring that the right environment is used for each test.
5. **Performance testing:** Machine learning can be used to analyze performance data and predict potential bottlenecks or issues, enabling testers to proactively address performance concerns.
6. **Log Analysis:** ML techniques can analyze logs generated during testing to identify patterns, trends, and anomalies that may indicate potential issues, facilitating faster root cause analysis and debugging.

## Real-life Examples of Machine Learning in Software Testing

**Facebook's Sapienz**
Facebook has developed an automated software testing tool called Sapienz, which leverages machine learning to identify and prioritize test cases. This tool has been instrumental in reducing the number of crashes in Facebook's Android app by 80%.

**Microsoft's DeepCode**
Microsoft has acquired DeepCode, a platform that uses machine learning to analyze source code and identify potential security vulnerabilities, bugs, and other issues. By incorporating DeepCode into their software

testing process, Microsoft can proactively address potential issues and improve overall software quality.

**Q2**

## Role of Software Test Engineer in Agile Mindset

Agile methodology works on continuous iteration of development and testing in the SDLC where development and testing activities are carried out in parallel. In Agile, testing activities begin at the start of the SDLC till the end of the process. It helps to develop workable products in fixed short time iterations from 1 to 2 weeks for the customer to get the immediate feedback. A tester's role in an Agile team includes following activities.

1. **Collaboration:** Test Engineers are involved in the development process from the beginning, collaborating closely with developers, product managers, and other stakeholders. They participate in requirements analysis and user story creation to ensure that to ensure that testing is aligned with the Agile sprint goals.
2. **Test Planning and Strategy:** Test Engineers are responsible to create test plans and strategies that align with Agile principles. They prioritize tests based on risk and business value, focusing on frequent, iterative testing to catch issues early in the development process.
3. **Continuous Testing:** In Agile development, testing is continuous throughout the development lifecycle. Test Engineers design and execute various types of tests, including unit tests, integration tests, regression tests, and acceptance tests, to validate each increment of functionality as it is developed or changed.
4. **Test Automation:** Test Engineers implement automation scripts to speed up testing in Agile development and reduce the manual testing efforts to get maximum benefits in quick time.

5. **Feedback Loop:** Test Engineers play a crucial role in providing feedback to the development team about the quality of the software. They report defects, track issues, and work collaboratively with developers to resolve problems quickly.
6. **Adaptability:** Agile environments are dynamic, with requirements and priorities evolving frequently. Test Engineers must be adaptable and responsive to change, adjusting test plans and strategies as needed to accommodate shifting priorities and requirements.
7. **Participate Proactively:** Agile testers should proactively participate in sprint groom meetings, daily stand-up, review, and retrospective meetings to improve the process and deliver quality product within time and according to customer needs.
8. **Root Cause Analysis:** Root cause analysis is the most important step in agile testing. The tester should be able to understand the root cause of an error, reproduction steps and possible solutions to this problem may help the developers to fix the issue and speeds up the development process.

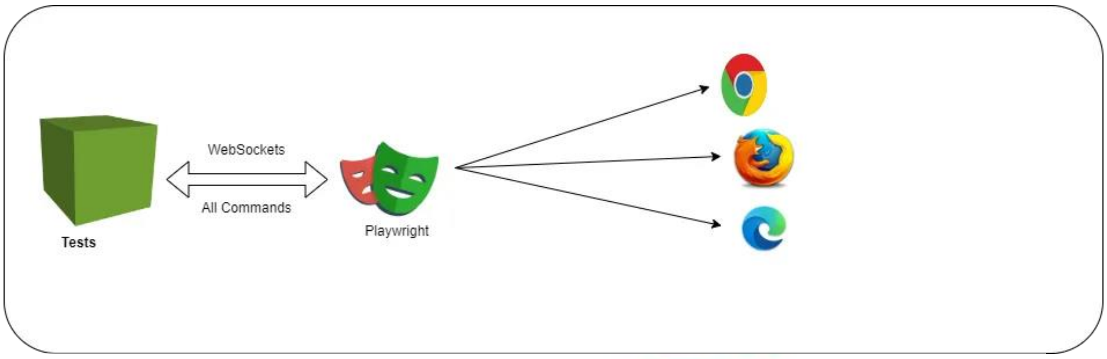
Overall, the role of a Test Engineer in an Agile mindset is to ensure that quality is built into every aspect of the software development process, fostering a culture of collaboration, feedback, and continuous improvement.

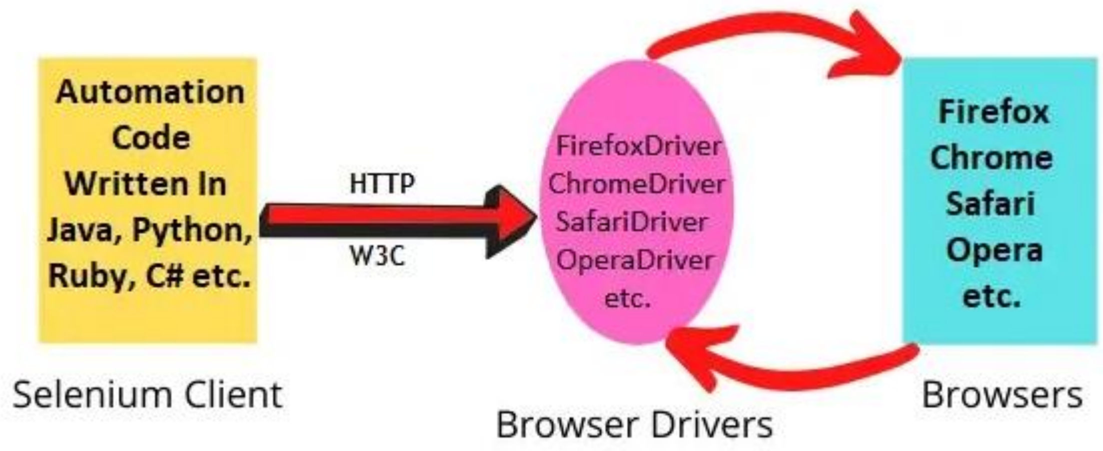
## Q3

## Playwright vs Selenium all def and architecture Differences, which app is suitable for what

| **Selenium** | **Playwright** |
|---|---|
| • **Architecture**: Selenium works on http protocol. Selenium uses the WebDriver API to interact between web browsers and browser drivers. | • **Architecture**: Playwright works on WebSocket protocol. This stays open for the duration of the test, so everything is sent on one connection. |

| | |
|---|---|
| Selenium is testing slower than playwright in performance and speed | Playwright is testing faster than selenium in performance and speed |
| Selenium requires separate WebDriver for each browser | Playwright provides built in drivers |
| Playwright supports all kind browsers Chromium, Firefox and Webkit through single API | Selenium has limited support for all kind of browsers Chrome, Firefox Edge. |
| Playwright has built-in auto-waiting functionality | Selenium requires explicit wait statements. |





**Selenium Architecture**

## Automation Steps

1. **Initialize your element / control**
2. **Locate element / control**
3. **Perform Action on element/control**
4. **Assert / Validate your action**

## Q4

## BDD Scenario Based case Study

## Feature File

```
Feature: Validate the Successfull login in Swag Lab

A short summary of the feature

@FunctionalTest
Scenario: Validate the Successfull login in Swag Lab with valid credentials
	Given Go to the URL https://www.saucedemo.com/
	And Enter user name
	And Enter password
	When Click login button for successfull Login
	Then User logged in successfully

@functionalTest
Scenario: Unsuccessful login with invalid credentials

   Given Go to the URL https://www.saucedemo.com/

   And Enter invalid username

   And Enter invalid password

   When Click login button

   Then User should see an error message
```

## Step Defination File

```
using System;

using TechTalk.SpecFlow;

using OpenQA.Selenium;

using OpenQA.Selenium.Chrome;

using NUnit.Framework;
```

```
namespace SwagLab_BDD.StepDefinitions
{
  [Binding]
  public class ValidateTheSuccessfullLoginInSwagLabStepDefinitions
  {
    ChromeDriver driver = new ChromeDriver();
    string url;
    [Given(@"Go to the URL https://www\.saucedemo\.com/")]
    public void GivenGoToTheURLHttpsWww_Saucedemo_Com()
    {
      driver.Manage().Window.Maximize();
      driver.Navigate().GoToUrl("https://www.saucedemo.com/");
    }

    [Given(@"Enter user name")]
    public void GivenEnterUserName()
    {
      driver.FindElement(By.Id("user-name")).SendKeys("standard_user");
    }

    [Given(@"Enter password")]
    public void GivenEnterPassword()
    {
      driver.FindElement(By.Id("password")).SendKeys("secret_sauce");
    }

    [When(@"Click login button for successfull Login")]
    public void WhenClickLoginButtonForSuccessfullLogin()
    {
      driver.FindElement(By.Id("login-button")).Click();
    }
```

```
        [Then(@"User logged in successfully")]
        public void ThenUserLoggedInSuccessfully()
        {
            string text = driver.FindElement(By.XPath("//input[@id='login-button']")).Text;
            Assert.AreEqual("Products", text);
        }
    }
}
```

## Marks Calculation BDD

## Feature File

```
Feature: Marks Calculation

Scenario: Calculate the average marks of different subjects
    Given I have marks for subject1 as 90
    And I have marks for subject2 as 85
    And I have marks for subject3 as 92
    When I calculate the average marks
    Then The average marks should be 89

Scenario: Calculate the total marks of different subjects
    Given I have marks for subject1 as 90
    And I have marks for subject2 as 85
    And I have marks for subject3 as 92
    When I calculate the total marks
    Then The total marks should be 267

Scenario: Calculate the highest marks among different subjects
    Given I have marks for subject1 as 90
```

```
  And I have marks for subject2 as 85
  And I have marks for subject3 as 92
  When I calculate the highest marks
  Then The highest marks should be 92
```

## Step Definition File

```
using TechTalk.SpecFlow;
using NUnit.Framework;

namespace MarksCalculationBDD.StepDefinitions
{
  [Binding]
  public class MarksCalculationSteps
  {
    private int subject1Marks;
    private int subject2Marks;
    private int subject3Marks;
    private double result;

    [Given(@"I have marks for subject1 as (\d+)")]
    public void GivenIHaveMarksForSubject1As(int marks)
    {
      subject1Marks = marks;
    }

    [Given(@"I have marks for subject2 as (\d+)")]
    public void GivenIHaveMarksForSubject2As(int marks)
    {
      subject2Marks = marks;
    }

    [Given(@"I have marks for subject3 as (\d+)")]
    public void GivenIHaveMarksForSubject3As(int marks)
    {
      subject3Marks = marks;
    }

    [When(@"I calculate the average marks")]
    public void WhenICalculateTheAverageMarks()
    {
      result = (subject1Marks + subject2Marks + subject3Marks) / 3.0;
    }

    [When(@"I calculate the total marks")]
    public void WhenICalculateTheTotalMarks()
    {
      result = subject1Marks + subject2Marks + subject3Marks;
    }

    [When(@"I calculate the highest marks")]
    public void WhenICalculateTheHighestMarks()
    {
      result = subject1Marks;
      if (subject2Marks > result)
         result = subject2Marks;
      if (subject3Marks > result)
         result = subject3Marks;
    }
```

```
        [Then(@"The average marks should be (\d+)")]
        public void ThenTheAverageMarksShouldBe(double expectedAverage)
        {
            Assert.AreEqual(expectedAverage, result);
        }

        [Then(@"The total marks should be (\d+)")]
        public void ThenTheTotalMarksShouldBe(int expectedTotal)
        {
            Assert.AreEqual(expectedTotal, result);
        }

        [Then(@"The highest marks should be (\d+)")]
        public void ThenTheHighestMarksShouldBe(int expectedHighest)
        {
            Assert.AreEqual(expectedHighest, result);
        }
    }
}
```

## Q5

### Page Object Model

### Page Class Code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using OpenQA.Selenium;
using OpenQA.Selenium.Chrome;
namespace SwagLab_AST_FAST_POM
{
  public class LoginPageClass :BaseClass
  {
    string actualText = "Products";
    public void LoginHelper(string user, string passcode)
    {
      chromeDriver.FindElement(By.Id(Locators.UserName)).SendKeys(user);
      chromeDriver.FindElement(By.Id(Locators.Password)).SendKeys(passcode);
      chromeDriver.FindElement(By.Id(Locators.Login)).Click();


    }


    public void SuccessfullMessageValidation()
    {
      string validationText =
chromeDriver.FindElement(By.XPath(Locators.ProductText)).Text;
      Assert.AreEqual(actualText, validationText);
    }
  }
}
```

## Login Test Class

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;


namespace SwagLab_AST_FAST_POM
```

```
{
    public class LoginTestCases
    {
        LoginPageClass loginPageClass = new LoginPageClass();
        public void LoginTest1()
        {
            loginPageClass.DriverInitialize();
            loginPageClass.OpenBrowserAndUrl();
            loginPageClass.LoginHelper("standard_user", "secret_sauce");
            loginPageClass.SuccessfullMessageValidation();

        }
    }
}
```

## Base Class

```
using OpenQA.Selenium.Chrome;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SwagLab_AST_FAST_POM
{
    public class BaseClass
    {
        public static ChromeDriver chromeDriver;
        public string url = "https://www.saucedemo.com/";

        /// <summary>
        /// Chrome Driver Initilaize
        /// </summary>
        public void DriverInitialize()


        {
            chromeDriver = new ChromeDriver();


        }


        public void CloseBrowserInsitance()
        {
            chromeDriver.Close();
            chromeDriver.Dispose();
        }


        public void OpenBrowserAndUrl()
        {
            chromeDriver.Manage().Window.Maximize();
            chromeDriver.Navigate().GoToUrl(url);


        }


    }
}
```

## Locator Class

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace SwagLab_AST_FAST_POM
{
  public static class Locators
  {
    public const string UserName = "user-name";
    public const string Password = "password";
    public const string Login = "password";
    public const string  ProductText = "password";
    public const string BagPackProduct = "add-to-cart-sauce-labs-backpack";
    public const string CartLink = "//a[@class='shopping_cart_link']";
  }
}
```

## Q6

### Error guessing technique

The main purpose of this technique is to identify common errors at any level of testing by exercising the following tasks:

- Enter blank space into the text fields.
- Null pointer exception.
- Enter invalid parameters.
- Divide by zero.
- Use maximum limit of files to be uploaded.
- Check buttons without entering values.

## Examples of Error guessing method

### Example1

A function of the application requires a mobile number which must be of 10 characters. Now, below are the techniques that can be applied to guess error in the mobile number field:

- What will be the result, if the entered character is other than a number?
- What will be the result, if entered characters are less than 10 digits?
- What will be the result, if the mobile field is left blank?

After implementing these techniques, if the output is similar to the expected result, the function is considered to be bug-free, but if the output is not similar to the expected result, so it is sent to the development team to fix the defects.

However, error guessing is the key technique among all testing techniques as it depends on the experience of a tester, but it does not give surety of highest quality benchmark. It does not provide full coverage to the software. This technique can yield a better result if combined with other techniques of testing.

1. **What will be the result if the entered character is other than a number?**

    - If the entered character is not a number (e.g., alphabets, special characters), it should ideally result in an error or validation message indicating that only numerical characters are allowed.

2. **What will be the result if entered characters are less than 10 digits?**

    - If the entered mobile number has fewer than 10 digits, it should result in an error or validation message indicating that the mobile number is incomplete and must be 10 digits long.

3. **What will be the result if the mobile field is left blank?**

    - If the mobile field is left blank, it should result in an error or validation message indicating that the mobile number is required and cannot be empty.

4. Example2
5. Suppose we have one bank account, and we have to deposit some money over there, but the amount will be accepted on a particular range of **which is 5000-7000**. So here, we will provide the different input's value until it covers the maximum test coverage based on the error guessing technique, and see whether it is accepted or give the error message:

| value | description |
|---|---|
| 6000 | Accept |
| 5555 | Accept |
| 4000 | Error message |

|  |  |
|---|---|
| 8000 | Error message |
| blank | Error message |
| 100$ | Error message |
| ---- | ---- |

Consider a scenario where you are testing the functionality of funds transfer in a banking application. There is a field where you need to enter the amount. The requirements say that you can transfer any amount between 100 and 100000

If you apply standard techniques of Boundary value analysis and Equivalence Partitioning, you may come up with the below validations:

99, 100, 101, 500000, 99999, 100000, 100001 -1000.

However, an experienced tester has seen that a few of the similar applications have not handled the cases with a negative value. He would use this experience to create error-guessing scenarios for the current application

## Some Common Error guessing Examples

1. **Email Validation**:
    - Invalid email format: The application should display an error message indicating that the email format is invalid.
    - Blank email field: The application should display an error message indicating that the email field is required.
    - Exceeding maximum length: The application should display an error message indicating that the email cannot exceed a certain length.
2. **Password Validation**:
    - Short or long password: The application should display an error message indicating that the password length does not meet the requirements.
    - Complexity requirements not met: The application should display an error message indicating that the password does not meet the complexity requirements.

- Blank password field: The application should display an error message indicating that the password field is required.

3. **Date Input**:

- Invalid date format: The application should display an error message indicating that the date format is invalid.
- Future date entered: The application should display an error message indicating that the date must be in the past.
- Blank date field: The application should display an error message indicating that the date field is required.

4. **Numeric Input**:

- Non-numeric characters entered: The application should display an error message indicating that only numeric characters are allowed.
- Negative number entered: The application should display an error message indicating that only positive numbers are allowed.
- Blank numeric field: The application should display an error message indicating that the numeric field is required.

5. **File Upload**:

- Unsupported file format: The application should display an error message indicating that the file format is not supported.
- File size exceeds limit: The application should display an error message indicating that the file size exceeds the maximum allowed.
- No file selected: The application should display an error message indicating that a file must be selected for upload.

6. **Dropdown Selection**:

- Invalid option selected: The application should display an error message indicating that the selected option is invalid.
- No option selected: The application should display an error message indicating that an option must be selected.
- Dropdown menu disabled or not visible: The application should ensure that the dropdown menu is enabled and visible for selection.

7. **Checkbox/Radio Button Selection**:

- None selected in checkboxes: The application should display an error message indicating that at least one checkbox must be selected.
- Invalid option selected in radio buttons: The application should display an error message indicating that the selected option is invalid.
- Checkbox or radio button disabled: The application should ensure that the checkbox or radio button is enabled for selection.

**Q7**

**Simple Case Study you are a Test Engineer Ad-Hoc System is already implemented what startegy do we need to implement do we need to have automation many bugs**

**which tool you will use**

**Scenario:**

You are a Test Engineer working for an e-commerce company that has recently implemented a new online shopping platform called "ShopNow." The platform allows users to browse products, add them to their cart, and make purchases online. As the Test Engineer, you are responsible for ensuring the quality and reliability of the ShopNow platform.

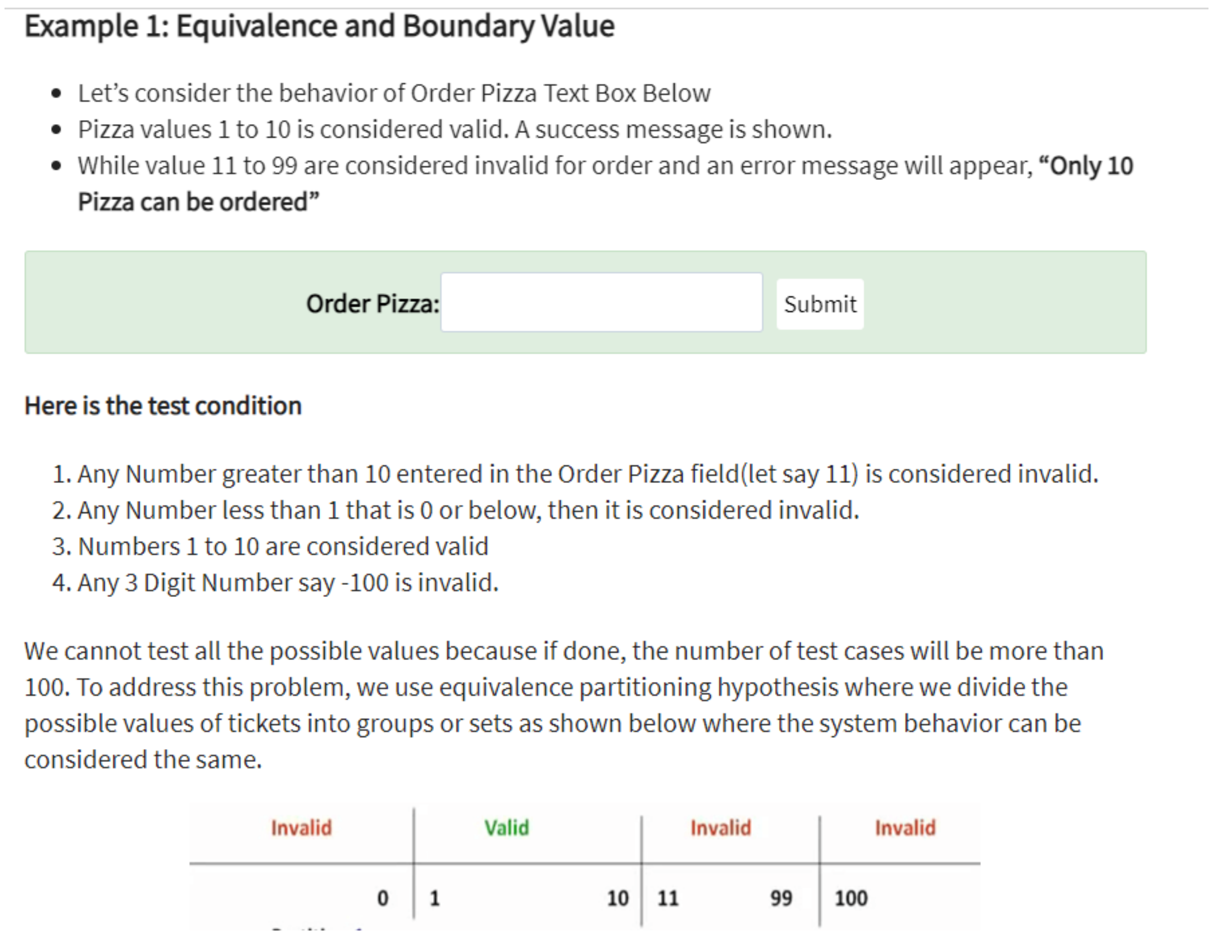
1. **Initial Assessment:**
   - Reviewing requirements documentation and technical specifications is crucial.
   - Understand the platform's features, functionalities, and user flows.
   - Identifying key features (user registration, product browsing, cart management, checkout process) is essential for targeted testing.
   - Understanding the target audience and usage scenarios helps tailor your testing efforts effectively.
2. **Test Planning:**
   - A detailed test plan is essential. It should cover the testing approach, objectives, scope, resources, and timelines.
3. **Risk Analysis:**
   - Prioritize testing based on risk to focus on critical functionalities first.
   - Identifying potential risks (security, performance, compatibility) is essential.
   - Consider different devices, browsers, and user environments.
4. **Automation Consideration:**
   - Focus on repetitive and time-consuming test cases.
   - To cover a wide range of scenarios quickly.
   - **Scenarios**:
   - **Login/Logout**: Automate user login and logout.
   - **Product Search**: Automate product search and verify results.
   - **Checkout Flow**: Automate end-to-end checkout process.
5. **Tools Selection**:
   - **UI Testing**: Consider tools like **Selenium WebDriver**, **Cypress**, or **TestCafe**.
   - **API Testing**: Use tools like **Postman**, **RestAssured**, or **SoapUI** for API testing.
6. **Bug Management**:

**Purpose**: Efficiently track and manage defects.

- **Bug Reporting**: Document defects with clear steps to reproduce.
- **Severity and Priority**: Prioritize based on impact (critical, major, minor).
-

**Q7**

**Boundary Value Analysis (Pizza Order, Login cases, Above, Below Minimum) how to implement boundary value on certain test cases**



## Example 1: Equivalence and Boundary Value

- Let's consider the behavior of Order Pizza Text Box Below
- Pizza values 1 to 10 is considered valid. A success message is shown.
- While value 11 to 99 are considered invalid for order and an error message will appear, **"Only 10 Pizza can be ordered"**

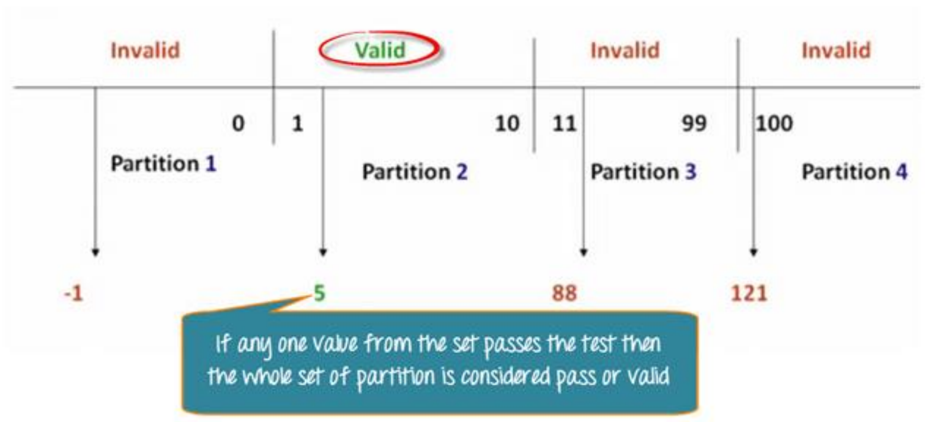
Order Pizza: Submit

**Here is the test condition**

1. Any Number greater than 10 entered in the Order Pizza field(let say 11) is considered invalid.
2. Any Number less than 1 that is 0 or below, then it is considered invalid.
3. Numbers 1 to 10 are considered valid
4. Any 3 Digit Number say -100 is invalid.

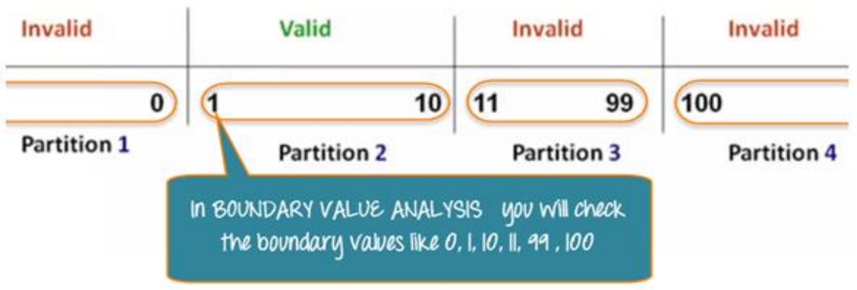
We cannot test all the possible values because if done, the number of test cases will be more than 100. To address this problem, we use equivalence partitioning hypothesis where we divide the possible values of tickets into groups or sets as shown below where the system behavior can be considered the same.

| Invalid | Valid | Invalid | Invalid |
|---|---|---|---|
| 0 | 1 10 | 11 99 | 100 |



Invalid
Valid
Invalid
Invalid
0
1
10
11
99
100
Partition 1
Partition 2
Partition 3
Partition 4
-1
5
88
121
If any one value from the set passes the test then the whole set of partition is considered pass or valid

**oundary Value Analysis**– in Boundary Value Analysis, you test boundaries between equivalence artitions

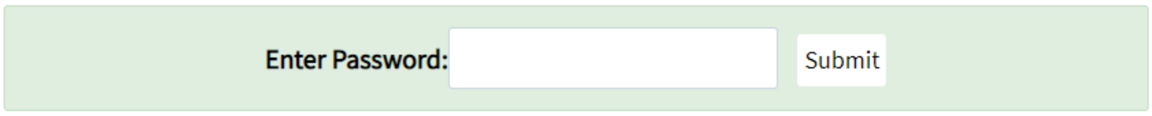


Invalid
Valid
Invalid
Invalid
0
1
10
11
99
100
Partition 1
Partition 2
Partition 3
Partition 4
In BOUNDARY VALUE ANALYSIS you will check the boundary values like 0, 1, 10, 11, 99, 100

## Example 2: Equivalence and Boundary Value

Following password field accepts minimum 6 characters and maximum 10 characters

That means results for values in partitions 0-5, 6-10, 11-14 should be equivalent



| Test Scenario # | Test Scenario Description | Expected Outcome |
|---|---|---|
| 1 | Enter 0 to 5 characters in password field | System should not accept |
| 2 | Enter 6 to 10 characters in password field | System should accept |
| 3 | Enter 11 to 14 character in password field | System should not accept |

## Examples 3: Input Box should accept the Number 1 to 10

Here we will see the Boundary Value Test Cases

| Test Scenario Description | Expected Outcome |
|---|---|
| Boundary Value = 0 | System should NOT accept |
| Boundary Value = 1 | System should accept |
| Boundary Value = 2 | System should accept |
| Boundary Value = 9 | System should accept |
| Boundary Value = 10 | System should accept |
| Boundary Value = 11 | System should NOT accept |

## Test Case Number #1

Let us assume a test case that takes the value of age from 21 to 65.

| BOUNDARY VALUE TEST CASE | | |
|---|---|---|
| **INVALID TEST CASE**<br><br>**(Min Value – 1)** | **VALID TEST CASES**<br><br>**(Min, +Min, Max, -Max)** | **INVALID TEST CASE**<br><br>**(Max Value + 1)** |
| **20** | **21, 22, 65, 64** | **66** |

From the above table, we can view the following inputs that are given.

- The minimum boundary value is given as 21.
- The maximum boundary value is given as 65.
- The valid inputs for testing purposes are 21, 22, 64 and 65.

## Example 1: Printer Copies

Suppose a printer needs to make and deliver printed copies, and the acceptable range is from 1 to 150. Here are the boundary values:

- **Minimum Value**: 1 (valid)
- **Maximum Value**: 150 (valid)
- **Invalid Values**:
    - Below Minimum: 0
    - Above Maximum: 151

We'll create four boundary value test cases:

1. **Valid Test Case**: 1 copy (minimum value)
2. **Valid Test Case**: 150 copies (maximum value)
3. **Invalid Test Case**: 0 copies (below minimum)
4. **Invalid Test Case**: 151 copies (above maximum)

## Example 2: 5-Digit Field

Consider a field that holds a maximum of 5 digits. The valid range is from 10000 to 99999. Here are the boundary values:

- **Minimum Value**: 10000 (valid)
- **Maximum Value**: 99999 (valid)
- **Invalid Values**:
    - Below Minimum: 9999
    - Above Maximum: 100000

We'll create four boundary value test cases:

1. **Valid Test Case**: 10000 (minimum value)
2. **Valid Test Case**: 99999 (maximum value)
3. **Invalid Test Case**: 9999 (below minimum)
4. **Invalid Test Case**: 100000 (above maximum)

By testing these boundary values, we ensure that the system handles extreme cases correctly.