SRM Institute of Science and Technology

Department of Computer Applications

Delhi – Meerut Road, Sikri Kalan, Ghaziabad, Uttar Pradesh – 201204

Circular – 2020-21

MCA 2nd semester
SOFTWARE TESTING (PCA20D06J)

Lab Manual

Lab 1: Test Case Design for Arithmetic Calculations

Title

Test Case Design for Arithmetic Calculations

Aim

To understand and apply various test case design techniques for arithmetic calculations to ensure their correctness and robustness.

Procedure

1. **Understand Requirements:** Analyze the requirements for arithmetic calculations (e.g., addition, subtraction, multiplication, division, handling of zero, negative numbers, large numbers).
2. **Identify Test Scenarios:** Based on the requirements, identify different scenarios that need to be tested.
3. **Apply Test Case Design Techniques:**
    - **Equivalence Partitioning:** Divide input data into valid and invalid equivalence classes.
    - **Boundary Value Analysis:** Select test cases at the boundaries of the input domains.
    - **Error Guessing:** Anticipate common errors (e.g., division by zero, overflow).
4. **Document Test Cases:** For each identified scenario, create a test case with a unique ID, description, preconditions, input data, expected output, and post-conditions.
5. **Execute Test Cases (Conceptual):** Mentally or conceptually execute the test cases to verify their effectiveness.

Source Code

```
// Placeholder for source code related to arithmetic calculations (e.g., a
simple calculator program)
// Example:
// public class Calculator {
//     public int add(int a, int b) { return a + b; }
//     public int subtract(int a, int b) { return a - b; }
//     public int multiply(int a, int b) { return a * b; }
//     public double divide(int a, int b) {
//         if (b == 0) throw new IllegalArgumentException("Cannot divide by
zero");
//         return (double) a / b;
//     }
// }
```

Input

```
// Placeholder for specific input values for test cases
// Example:
// Test Case 1 (Addition - Positive Numbers): a=5, b=3
// Test Case 2 (Subtraction - Negative Result): a=2, b=7
// Test Case 3 (Division - By Zero): a=10, b=0
```

Expected Output

```
// Placeholder for expected output for each test case
// Example:
// Test Case 1: 8
// Test Case 2: -5
// Test Case 3: Error: Cannot divide by zero
```

Lab 2: Test Case Report for Sorting of n number

Title

Test Case Report for Sorting of n number

Aim

To design and document test cases for a program that sorts 'n' numbers and to prepare a comprehensive test case report.

Procedure

1. **Understand Requirements:** Analyze the requirements for a sorting program (e.g., ascending/descending order, handling positive/negative numbers, duplicates, empty list, single element list, large number of elements).
2. **Identify Test Scenarios:** Determine various scenarios for sorting, including edge cases and invalid inputs.
3. **Design Test Cases:** Apply techniques like equivalence partitioning and boundary value analysis to create detailed test cases.
4. **Prepare Test Case Report:** Document all test cases in a structured report format, including:
    - Test Case ID
    - Test Case Description
    - Preconditions
    - Input Data
    - Expected Output
    - Actual Output (to be filled during execution)
    - Status (Pass/Fail)
    - Remarks

Source Code

```
// Placeholder for source code of a sorting program (e.g., Bubble Sort, Quick Sort)
// Example:
// public class Sorter {
//     public int[] bubbleSort(int[] arr) { /* ... sorting logic ... */ }
// }
```

Input

```
// Placeholder for specific input arrays for test cases
// Example:
// Test Case 1 (Positive numbers): [5, 2, 8, 1, 9]
// Test Case 2 (Negative numbers): [-3, -1, -5]
// Test Case 3 (Duplicates): [4, 2, 4, 1, 2]
// Test Case 4 (Empty array): []
```

Expected Output

```
// Placeholder for expected sorted arrays
// Example:
```

```
// Test Case 1: [1, 2, 5, 8, 9]
// Test Case 2: [-5, -3, -1]
// Test Case 3: [1, 2, 2, 4, 4]
// Test Case 4: []
```

Lab 3: Preparation of Test Case Report on Triangle Program

Title

Preparation of Test Case Report on Triangle Program

Aim

To design test cases for a program that determines the type of triangle (equilateral, isosceles, scalene, or invalid) given three side lengths, and to prepare a test case report.

Procedure

1. **Understand Requirements:** Analyze the rules for classifying triangles based on side lengths.
2. **Identify Test Scenarios:** Identify scenarios for valid triangles (equilateral, isosceles, scalene) and invalid triangle conditions (e.g., sum of two sides less than or equal to the third, zero or negative side lengths).
3. **Design Test Cases:** Use equivalence partitioning and boundary value analysis to create test cases covering all valid and invalid conditions.
4. **Prepare Test Case Report:** Document the test cases in a report format, including the standard fields (ID, description, input, expected output, etc.).

Source Code

```
// Placeholder for source code of a triangle classification program
// Example:
// public class TriangleClassifier {
//     public String classify(int a, int b, int c) { /* ... classification logic
... */ }
// }
```

Input

```
// Placeholder for specific side lengths for test cases
// Example:
// Test Case 1 (Equilateral): a=5, b=5, c=5
// Test Case 2 (Isosceles): a=4, b=4, c=6
// Test Case 3 (Scalene): a=3, b=4, c=5
// Test Case 4 (Invalid - Sum of two sides): a=1, b=2, c=5
// Test Case 5 (Invalid - Zero side): a=0, b=3, c=4
```

Expected Output

```
// Placeholder for expected triangle type or error message
// Example:
// Test Case 1: Equilateral
// Test Case 2: Isosceles
// Test Case 3: Scalene
// Test Case 4: Invalid Triangle
// Test Case 5: Invalid Input: Side length cannot be zero or negative
```

Lab 4: Preparation of Test Case Report on Binary Search Program

Title

Preparation of Test Case Report on Binary Search Program

Aim

To design test cases for a binary search program and to prepare a comprehensive test case report.

Procedure

1. **Understand Requirements:** Analyze the binary search algorithm, including its prerequisites (sorted array) and behavior for found/not found elements.
2. **Identify Test Scenarios:** Identify scenarios such as element found (first, last, middle), element not found, empty array, single element array, array with duplicates, and large arrays.
3. **Design Test Cases:** Apply test case design techniques to cover all identified scenarios.
4. **Prepare Test Case Report:** Document the test cases in a standard report format.

Source Code

```
// Placeholder for source code of a binary search program
// Example:
// public class BinarySearch {
//     public int search(int[] arr, int target) { /* ... binary search logic ...
*/ }
// }
```

Input

```
// Placeholder for specific arrays and target values for test cases
// Example:
// Test Case 1 (Element found - middle): arr=[1, 3, 5, 7, 9], target=5
// Test Case 2 (Element not found): arr=[1, 3, 5, 7, 9], target=4
// Test Case 3 (Empty array): arr=[], target=10
// Test Case 4 (Single element - found): arr=[7], target=7
```

Expected Output

```
// Placeholder for expected index or not found indicator
// Example:
// Test Case 1: 2
// Test Case 2: -1 (or appropriate 'not found' indicator)
// Test Case 3: -1
// Test Case 4: 0
```

Lab 5: Develop a Login Form and Prepare Test Case Report

Title

Develop a Login Form and Prepare Test Case Report

Aim

To develop a simple login form and then design and prepare a test case report to verify its functionality and security.

Procedure

1. **Develop Login Form:** Create an HTML/CSS/JavaScript-based login form with fields for username/email and password, and a login button. Implement basic validation (e.g., non-empty fields).
2. **Understand Requirements:** Define valid and invalid login credentials, error messages, and redirection logic.
3. **Identify Test Scenarios:** Identify scenarios such as valid login, invalid username, invalid password, empty fields, SQL injection attempts (conceptual), and password strength validation (if applicable).
4. **Design Test Cases:** Create test cases covering functional and non-functional aspects (e.g., UI responsiveness, error message display).
5. **Prepare Test Case Report:** Document the test cases and their expected outcomes.

Source Code

```
<!-- <form id="loginForm">
    <label for="username">Username:</label>
    <input type="text" id="username" name="username">
    <label for="password">Password:</label>
    <password type="password" id="password" name="password">
    <button type="submit">Login</button>
    <div id="message"></div>
</form>
<script>
    document.getElementById('loginForm').addEventListener('submit',
function(event) {
        event.preventDefault();
        const username = document.getElementById('username').value;
        const password = document.getElementById('password').value;
        const messageDiv = document.getElementById('message');

        if (username === 'user' && password === 'pass') {
            messageDiv.textContent = 'Login Successful!';
            messageDiv.style.color = 'green';
        } else {
            messageDiv.textContent = 'Invalid Username or Password.';
            messageDiv.style.color = 'red';
        }
    });
</script> -->
```

Input

```
// Placeholder for specific login credentials for test cases
// Example:
// Test Case 1 (Valid): username="user", password="pass"
// Test Case 2 (Invalid Username): username="wrong", password="pass"
// Test Case 3 (Invalid Password): username="user", password="wrong"
// Test Case 4 (Empty Username): username="", password="pass"
```

Expected Output

```
// Placeholder for expected UI behavior and messages
// Example:
// Test Case 1: "Login Successful!" message, redirection to dashboard.
// Test Case 2: "Invalid Username or Password." message.
// Test Case 3: "Invalid Username or Password." message.
// Test Case 4: "Please enter username." message.
```

Lab 6: Develop a Student Mark sheet application and Conducting Testing

Title

Develop a Student Mark sheet application and Conducting Testing

Aim

To develop a student mark sheet application that calculates total marks and grades, and to conduct thorough testing by preparing a test case report.

Procedure

1. **Develop Mark Sheet Application:** Create an application (e.g., using a programming language like Python or Java, or a web-based interface) that allows input of student details and marks for various subjects, calculates total marks and percentage, and assigns grades based on predefined criteria.
2. **Understand Requirements:** Define grading criteria, input constraints (e.g., marks between 0-100), and error handling for invalid inputs.
3. **Identify Test Scenarios:** Identify scenarios for valid mark entries, invalid mark entries (e.g., negative, >100), boundary marks for grades, handling of missing data, and calculation accuracy.
4. **Design Test Cases:** Apply test case design techniques to create comprehensive test cases.
5. **Conduct Testing:** Execute the designed test cases on the developed application.
6. **Prepare Test Case Report:** Document the test cases, actual results, and status (Pass/Fail), along with any bugs found.

Source Code

```
// Placeholder for source code of a student mark sheet application
// Example (conceptual Python):
// def calculate_grade(total_marks):
//     if total_marks >= 90: return 'A'
//     elif total_marks >= 80: return 'B'
//     // ...
//     else: return 'F'
//
// def process_marksheet(student_name, marks_dict):
//     // ... calculate total, percentage, grade ...
```

Input

```
// Placeholder for specific student data and marks for test cases
// Example:
// Test Case 1 (All subjects high marks): Student A, Math=95, Science=90,
English=88
// Test Case 2 (Boundary for A grade): Student B, Math=89, Science=90,
English=91
// Test Case 3 (Invalid mark): Student C, Math=105, Science=70, English=60
// Test Case 4 (Fail case): Student D, Math=30, Science=25, English=35
```

Expected Output

```
// Placeholder for expected total marks, percentage, and grade
// Example:
// Test Case 1: Total=273, Percentage=91%, Grade='A'
// Test Case 2: Total=270, Percentage=90%, Grade='A'
// Test Case 3: Error: Marks must be between 0 and 100
// Test Case 4: Total=90, Percentage=30%, Grade='F'
```

Lab 7: Develop a Employee salary Processing application and Prepare Test Case Report

Title

Develop an Employee Salary Processing application and Prepare Test Case Report

Aim

To develop an application for processing employee salaries, including calculations for deductions and net pay, and to prepare a detailed test case report.

Procedure

1. **Develop Salary Processing Application:** Create an application that takes employee details (e.g., basic salary, allowances, deductions like tax, provident fund) and calculates gross pay, net pay, and displays a salary slip.
2. **Understand Requirements:** Define salary components, tax slabs, deduction rules, and error handling for invalid financial inputs.
3. **Identify Test Scenarios:** Identify scenarios for various salary ranges, different deduction configurations, boundary values for tax slabs, and invalid numerical inputs.
4. **Design Test Cases:** Apply test case design techniques to create comprehensive test cases covering all calculations and edge cases.
5. **Prepare Test Case Report:** Document the test cases, actual results, and status.

Source Code

```
// Placeholder for source code of an employee salary processing application
// Example (conceptual):
// class Employee {
//     double basicSalary;
//     double calculateGrossPay() { /* ... */ }
//     double calculateTax() { /* ... */ }
//     double calculateNetPay() { /* ... */ }
// }
```

Input

```
// Placeholder for specific employee data and salary components for test cases
// Example:
// Test Case 1 (Standard salary): Basic=50000, Allowances=10000, Deductions=5000
// Test Case 2 (High salary, different tax slab): Basic=150000,
Allowances=20000, Deductions=10000
// Test Case 3 (Zero basic salary): Basic=0, Allowances=0, Deductions=0
// Test Case 4 (Negative deduction): Basic=40000, Allowances=5000, Deductions=-
1000
```

Expected Output

```
// Placeholder for expected gross pay, net pay, and other calculated values
// Example:
// Test Case 1: Gross Pay=60000, Net Pay=55000
```

```
// Test Case 2: Gross Pay=170000, Net Pay=... (after tax calculation)
// Test Case 3: Gross Pay=0, Net Pay=0
// Test Case 4: Error: Deduction cannot be negative
```

Lab 8: Develop a Flight Reservation application and Prepare Test Case Report

Title

Develop a Flight Reservation application and Prepare Test Case Report

Aim

To develop a simplified flight reservation application and to prepare a comprehensive test case report to ensure its functionality, booking process, and error handling.

Procedure

1. **Develop Flight Reservation Application:** Create an application that allows users to search for flights (origin, destination, date), select a flight, enter passenger details, and simulate a booking process.
2. **Understand Requirements:** Define search criteria, booking flow, seat availability, passenger information validation, and confirmation/error messages.
3. **Identify Test Scenarios:** Identify scenarios such as valid flight search, no flights found, selecting a flight, entering valid/invalid passenger details, attempting to book an unavailable seat, and handling payment errors (conceptual).
4. **Design Test Cases:** Apply test case design techniques to cover the entire booking flow and various error conditions.
5. **Prepare Test Case Report:** Document the test cases, actual results, and status.

Source Code

```
// Placeholder for source code of a flight reservation application
// Example (conceptual):
// class FlightSearch {
//     List<Flight> searchFlights(String origin, String destination, Date date)
{ /* ... */ }
// }
// class Booking {
//     boolean bookFlight(Flight selectedFlight, Passenger passenger) { /* ...
*/ }
// }
```

Input

```
// Placeholder for specific search queries and passenger details for test cases
// Example:
// Test Case 1 (Valid search): Origin="DEL", Destination="MUM", Date="2025-12-
25"
// Test Case 2 (No flights): Origin="XYZ", Destination="ABC", Date="2025-12-25"
// Test Case 3 (Booking with valid details): Flight_ID="AI101",
Passenger_Name="John Doe", Age=30
// Test Case 4 (Booking with invalid age): Flight_ID="AI101",
Passenger_Name="Jane Doe", Age=0
```

Expected Output

```
// Placeholder for expected search results, booking confirmations, or error
messages
// Example:
// Test Case 1: List of available flights matching criteria.
// Test Case 2: "No flights found for this route." message.
// Test Case 3: "Booking successful! Confirmation ID: XYZ123"
// Test Case 4: "Invalid Age. Please enter a valid age."
```

Lab 9: Web site Testing

Title

Web site Testing

Aim

To perform comprehensive testing on a given website, covering functional, usability, performance, and security aspects, and to document the findings.

Procedure

1. **Select a Website:** Choose a publicly accessible website for testing (e.g., an e-commerce site, a news portal, a social media platform).
2. **Understand Website Functionality:** Explore the website to understand its features, navigation, and user flows.
3. **Identify Test Categories:**
    - **Functional Testing:** Verify all links, forms, buttons, search functionality, and data submission work as expected.
    - **Usability Testing:** Assess ease of navigation, clarity of content, and overall user experience.
    - **Performance Testing (Basic):** Observe page load times, responsiveness under normal usage.
    - **Security Testing (Basic):** Check for common vulnerabilities like broken links, exposed sensitive information (not ethical hacking).
    - **Compatibility Testing:** Check responsiveness on different browsers and devices (if possible).
4. **Design Test Cases/Checklist:** Create a checklist or informal test cases for each identified category.
5. **Execute Tests:** Systematically go through the website and execute the planned tests.
6. **Document Findings:** Record observations, actual results, and any bugs or issues found in a report format.

Source Code

```
// N/A - This lab focuses on testing an existing website, not developing one.
// No source code is provided as the subject is an external website.
```

Input

```
// Placeholder for typical user interactions and data entered on the website
// Example:
// - Navigating to different pages
// - Submitting a contact form with valid/invalid data
// - Using the search bar with different keywords
// - Clicking on various links and buttons
// - Trying to register with existing/invalid email
```

Expected Output

```
// Placeholder for expected website behavior and responses
// Example:
// - Page loads correctly and quickly.
// - Form submission is successful, and a confirmation message is displayed.
// - Search results are relevant.
// - All links are functional and lead to the correct pages.
// - Error messages are displayed for invalid inputs.
```

Lab 10: Software Test Automation using testing tool

Title

Software Test Automation using Testing Tool

Aim

To understand the concepts of software test automation and to automate a set of test cases using a chosen testing tool (e.g., Selenium, QTP/UFT, TestComplete).

Procedure

1. **Choose a Testing Tool:** Select a suitable automation testing tool (e.g., Selenium for web applications).
2. **Identify Test Cases for Automation:** Select a few simple, repetitive test cases from previous labs or a small web application that are good candidates for automation.
3. **Learn Tool Basics:** Familiarize yourself with the basic functionalities of the chosen tool (e.g., element locators, actions, assertions in Selenium).
4. **Develop Automation Scripts:** Write automation scripts for the selected test cases.
5. **Execute Automation Scripts:** Run the developed scripts and observe the execution.
6. **Analyze Results:** Review the automation test reports generated by the tool.
7. **Document Automation Process:** Record the steps taken, the tool used, and the challenges faced.

Source Code

```
// Placeholder for automation scripts (e.g., Selenium WebDriver code in
Java/Python)
// Example (Selenium with Python):
// from selenium import webdriver
// from selenium.webdriver.common.by import By
//
// driver = webdriver.Chrome()
// driver.get("http://www.example.com/login")
// driver.find_element(By.ID, "username").send_keys("testuser")
// driver.find_element(By.ID, "password").send_keys("testpass")
// driver.find_element(By.ID, "loginButton").click()
// // Add assertions to verify login success
// driver.quit()
```

Input

```
// Placeholder for data used by automation scripts (e.g., URLs, login
credentials)
// Example:
// - URL: "http://www.example.com/login"
// - Username: "testuser"
// - Password: "testpass"
```

Expected Output

```
// Placeholder for expected outcomes of the automated tests (e.g., successful
login, element found)
// Example:
// - Login successful, redirected to dashboard.
// - Specific text element found on the page.
// - No errors during script execution.
```

Lab 11: Writing and Tracking Test Cases

Title

Writing and Tracking Test Cases

Aim

To gain practical experience in writing well-structured test cases and using a system (manual or simple tool) to track their execution status.

Procedure

1. **Select a Small Application/Feature:** Choose a simple application or a specific feature (e.g., a calculator, a basic form, or a small module from a previous lab).
2. **Identify Requirements:** Clearly understand the functionality to be tested.
3. **Write Test Cases:** For the selected application/feature, write a comprehensive set of test cases following a standard format (ID, Description, Preconditions, Steps, Input Data, Expected Output). Focus on clarity and completeness.
4. **Choose a Tracking Method:** Use a simple spreadsheet (Excel/Google Sheets) or a basic test management tool (if available) to track the test cases.
5. **Execute and Track:** Simulate the execution of each test case and update its status (e.g., Not Run, Pass, Fail, Blocked) in the tracking system.
6. **Document Findings:** Summarize the testing efforts, including the number of test cases written, executed, passed, and failed.

Source Code

```
// N/A - This lab focuses on the process of writing and tracking test cases, not on developing an application.
// Source code would be for the application being tested, which is not part of this lab's deliverable.
```

Input

```
// Placeholder for conceptual inputs used during test case execution
// Example:
// - Varies depending on the application/feature chosen for testing.
// - Could be values entered into forms, buttons clicked, navigation paths followed.
```

Expected Output

```
// Placeholder for conceptual expected outcomes of the test cases
// Example:
// - Varies depending on the application/feature chosen for testing.
// - Could be specific messages displayed, correct calculations, successful data submission.
```

Lab 12: Bug Tracking System

Title

Bug Tracking System

Aim

To understand the importance of bug tracking, learn how to use a bug tracking system, and practice reporting and managing defects.

Procedure

1. **Understand Bug Life Cycle:** Familiarize yourself with the typical bug life cycle (New, Assigned, Open, Fixed, Retest, Reopen, Closed, Deferred, Duplicate, Rejected).
2. **Choose a Bug Tracking System:** Use a simulated bug tracking system (e.g., a simple spreadsheet mimicking fields like Bug ID, Summary, Description, Steps to Reproduce, Expected Result, Actual Result, Severity, Priority, Status, Assigned To, Reported By, Date). If available, use a real tool like Jira (community edition), Bugzilla, or Redmine.
3. **Identify Bugs (Simulated/Actual):** From previous labs or a given application, identify a few (simulated or actual) bugs.
4. **Report Bugs:** For each identified bug, create a detailed bug report in the chosen system, ensuring all necessary fields are filled accurately.
5. **Track Bug Status:** Simulate the various stages of the bug life cycle by updating the status of the reported bugs.
6. **Document Learning:** Summarize the experience of using the bug tracking system and the benefits of defect management.

Source Code

```
// N/A - This lab focuses on using a bug tracking system, not on developing one
or an application.
// No source code is provided.
```

Input

```
// Placeholder for bug details entered into the tracking system
// Example:
// - Bug Summary: "Login button not clickable on mobile"
// - Steps to Reproduce: "1. Open login page on Chrome mobile. 2. Enter
credentials. 3. Try to click login button."
// - Expected Result: "Login button should be clickable and initiate login."
// - Actual Result: "Login button is unresponsive."
// - Severity: High, Priority: High
```

Expected Output

```
// Placeholder for the structured bug report entries within the tracking system
// Example:
// - A new bug entry with all fields populated.
// - Status changes reflecting the bug's progress (e.g., from 'New' to
'Assigned' to 'Fixed').
```

Lab 13: Basic Operation of Selenium Testing tool

Title

Basic Operation of Selenium Testing Tool

Aim

To get hands-on experience with the fundamental operations of Selenium WebDriver for automating web browser interactions.

Procedure

1. **Setup Selenium Environment:** Install necessary components (e.g., Java/Python, Selenium WebDriver library, browser driver like ChromeDriver/GeckoDriver).
2. **Launch Browser:** Write a simple script to launch a web browser using Selenium WebDriver.
3. **Navigate to URL:** Write a script to navigate to a specific URL.
4. **Locate Elements:** Practice identifying web elements using various locators (ID, Name, Class Name, Tag Name, Link Text, Partial Link Text, CSS Selector, XPath).
5. **Perform Actions:** Perform basic actions on elements (e.g., `click()`, `send_keys()`, `clear()`).
6. **Get Element Properties:** Retrieve text, attributes, or CSS properties of elements.
7. **Close Browser:** Learn how to close the browser session.
8. **Document Scripts:** Keep a record of the scripts written and their purpose.

Source Code

```
# Placeholder for basic Selenium Python scripts
# Example:
from selenium import webdriver
from selenium.webdriver.common.by import By
import time

# Initialize the Chrome driver (ensure chromedriver is in your PATH or specify
path)
driver = webdriver.Chrome()

try:
    # Navigate to a website
    driver.get("https://www.google.com")
    print("Navigated to Google.")

    # Find the search box by name and type a query
    search_box = driver.find_element(By.NAME, "q")
    search_box.send_keys("Selenium WebDriver")
    print("Typed 'Selenium WebDriver' into search box.")

    # Find the search button by name and click it
    # Note: Google's search button might be tricky, using a common one or
submitting form
    # For demonstration, let's assume there's a button with name 'btnK'
    # Or simply submit the form
    search_box.submit()
    print("Submitted search query.")

    time.sleep(3) # Wait for results to load
```

```
    # Verify title of the page
    print(f"Page title after search: {driver.title}")

except Exception as e:
    print(f"An error occurred: {e}")

finally:
    # Close the browser
    driver.quit()
    print("Browser closed.")
```

Input

```
// Placeholder for URLs and data used in Selenium scripts
// Example:
// - URL: "https://www.google.com"
// - Search query: "Selenium WebDriver"
```

Expected Output

```
// Placeholder for expected console output and browser behavior
// Example:
// - Browser opens and navigates to Google.
// - "Selenium WebDriver" is typed into the search bar.
// - Search results page loads.
// - Console output showing page title or confirmation messages.
```

Lab 14: Working with Selenium Components

Title

Working with Selenium Components

Aim

To explore and utilize various advanced components and features of Selenium WebDriver, such as waits, dropdowns, alerts, and handling multiple windows/frames.

Procedure

1. **Implicit and Explicit Waits:** Implement different types of waits to handle dynamic web elements and synchronization issues.
2. **Handling Dropdowns:** Use `Select` class (in Java/Python) to interact with dropdown menus.
3. **Handling Alerts/Pop-ups:** Learn to accept, dismiss, and get text from JavaScript alerts.
4. **Handling Multiple Windows/Tabs:** Switch between different browser windows or tabs.
5. **Handling Frames:** Switch between different frames within a web page.
6. **Taking Screenshots:** Capture screenshots of the web page during test execution.
7. **Document Scripts:** Record the scripts and the specific Selenium components used.

Source Code

```
# Placeholder for Selenium Python scripts demonstrating various components
# Example:
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import Select
import time

driver = webdriver.Chrome()

try:
    driver.get("https://www.selenium.dev/selenium/web/dropdowns.html")

    # Handling Dropdown
    select_element = driver.find_element(By.ID, "dropdown")
    select = Select(select_element)
    select.select_by_visible_text("Option 2")
    print("Selected 'Option 2' from dropdown.")
    time.sleep(2)

    driver.get("https://www.selenium.dev/selenium/web/alerts.html")
    # Handling Alert
    driver.find_element(By.ID, "alert").click()
    WebDriverWait(driver, 10).until(EC.alert_is_present())
    alert = driver.switch_to.alert
    print(f"Alert text: {alert.text}")
    alert.accept()
    print("Alert accepted.")
    time.sleep(2)

    # Taking Screenshot
```

```
    driver.save_screenshot("screenshot_example.png")
    print("Screenshot taken: screenshot_example.png")

except Exception as e:
    print(f"An error occurred: {e}")

finally:
    driver.quit()
```

Input

```
// Placeholder for URLs and data used in Selenium scripts for component
interaction
// Example:
// - URL with dropdowns: "https://www.selenium.dev/selenium/web/dropdowns.html"
// - URL with alerts: "https://www.selenium.dev/selenium/web/alerts.html"
```

Expected Output

```
// Placeholder for expected console output and browser behavior
// Example:
// - Dropdown option selected.
// - Alert text printed to console, alert dismissed.
// - Screenshot file created.
```

Lab 15: Selenium Web driver Handling

Title

Selenium Web driver Handling

Aim

To gain a deeper understanding of advanced Selenium WebDriver concepts, including driver management, handling cookies, and executing JavaScript.

Procedure

1. **Driver Management:** Understand how to properly initialize and quit WebDriver instances, and manage browser profiles/options.
2. **Handling Cookies:** Add, delete, and get cookies using WebDriver.
3. **Executing JavaScript:** Execute JavaScript code directly from Selenium to interact with elements or retrieve information.
4. **Handling File Uploads (Conceptual):** Understand how to interact with file upload elements.
5. **Page Object Model (Conceptual):** Learn the basic principles of the Page Object Model design pattern for better test automation framework design.
6. **Error Handling in Automation:** Implement `try-except` blocks (Python) or `try-catch` (Java) for robust automation scripts.
7. **Document Scripts:** Record the scripts and the advanced WebDriver handling techniques used.

Source Code

```
# Placeholder for advanced Selenium Python scripts
# Example:
from selenium import webdriver
from selenium.webdriver.common.by import By
import time

driver = webdriver.Chrome()

try:
    driver.get("https://www.google.com")

    # Add a cookie
    driver.add_cookie({"name": "mycookie", "value": "testvalue"})
    print("Added a cookie.")

    # Get all cookies
    cookies = driver.get_cookies()
    print("Current cookies:", cookies)

    # Execute JavaScript
    driver.execute_script("alert('Hello from Selenium!');")
    print("Executed JavaScript alert.")
    time.sleep(2) # See the alert
    driver.switch_to.alert.accept() # Dismiss the alert
    print("Alert dismissed.")

    # Get page title using JavaScript
```

```
    js_title = driver.execute_script("return document.title;")
    print(f"Page title via JavaScript: {js_title}")

except Exception as e:
    print(f"An error occurred: {e}")

finally:
    driver.quit()
```

Input

```
// Placeholder for URLs and data used in advanced Selenium scripts
// Example:
// - Any URL for cookie manipulation or JavaScript execution.
```

Expected Output

```
// Placeholder for expected console output and browser behavior
// Example:
// - Cookie added and retrieved successfully.
// - JavaScript alert appears and is dismissed.
// - Page title retrieved via JavaScript is printed.
```