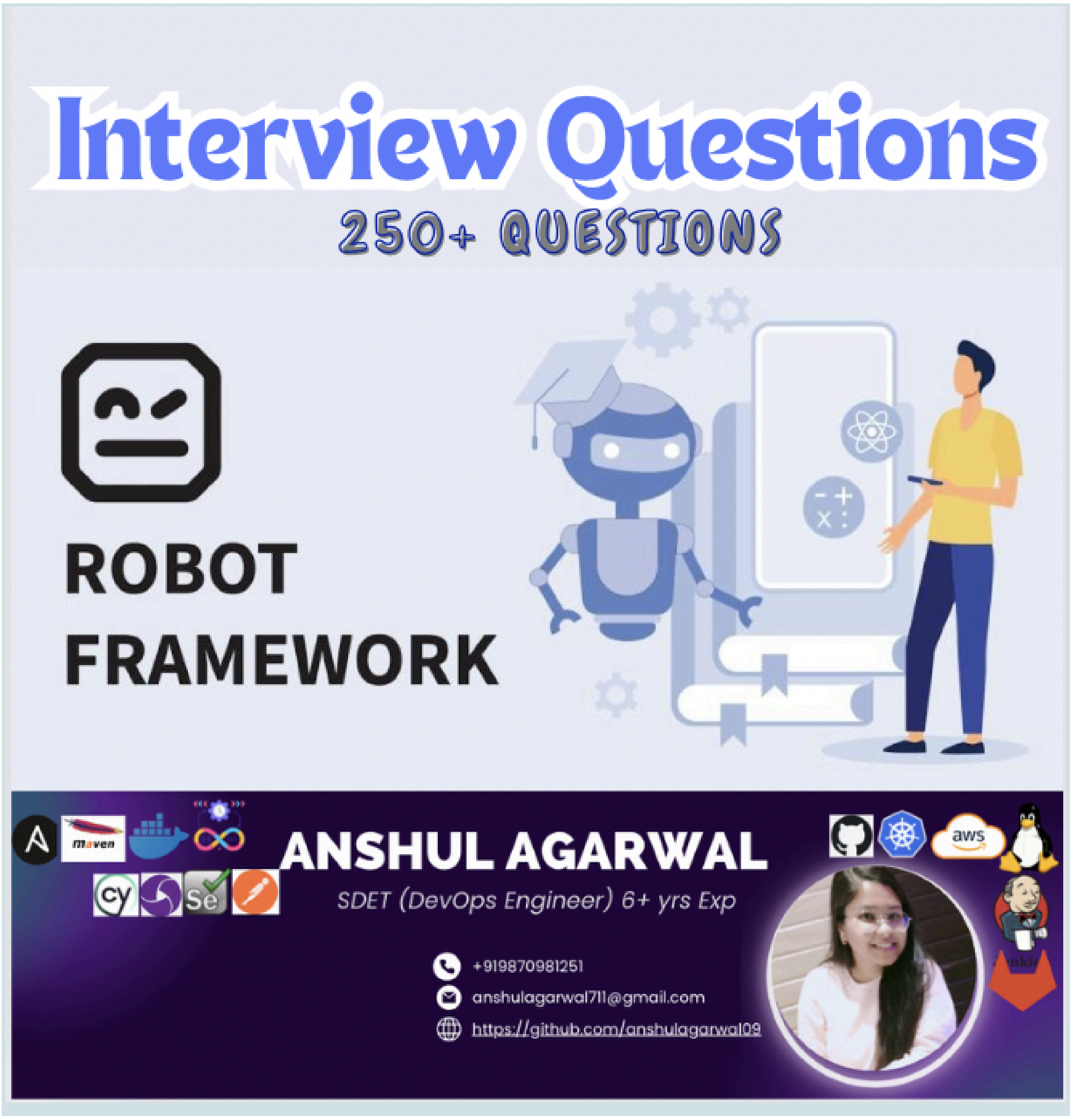
https://www.linkedin.com/in/anshulagarwal30/



Interview Questions

250+ QUESTIONS

ROBOT FRAMEWORK

ANSHUL AGARWAL

SDET (DevOps Engineer) 6+ yrs Exp

+919870981251

anshulagarwal711@gmail.com

https://github.com/anshulagarwal09

ROBOT FRAMEWORK Questions

1-20: Basics and Introduction

1. What is Robot Framework?
Robot Framework is an open-source automation framework used for test automation and robotic process automation (RPA). It is keyword-driven and uses tabular test data syntax, which makes it easy to read and write tests.

Robot Framework is an open-source test automation framework for acceptance testing and acceptance test-driven development (ATDD). It uses keyword-driven testing and has a highly extensible architecture.

2. What are the main components of Robot Framework?
The main components are:
- Test Data: Defines the test cases, suites, and keywords.
- Test Libraries: Provides the keywords (built-in, external, and custom libraries).
- Test Runner: Executes the test cases and generates reports (e.g., the `robot` command).

3. How do you install Robot Framework?
You can install Robot Framework using pip:
```sh
pip install robotframework
```

4. What is a test suite in Robot Framework?
A test suite is a collection of test cases. In Robot Framework, it is defined in a `.robot` file or a directory containing multiple `.robot` files.

5. What is a keyword in Robot Framework?
A keyword is a reusable action or a set of actions that can be used in test cases. Keywords can be user-defined or provided by libraries.

6. How do you create a user-defined keyword?
User-defined keywords are created in the `*** Keywords ***` section of a `.robot` file:
```robot
*** Keywords ***
Login To Application
    [Arguments]    ${username}    ${password}
    Input Text     id=username_field    ${username}
    Input Text     id=password_field    ${password}
    Click Button   id=login_button
```

7. What is a library in Robot Framework?

A library provides a set of keywords that can be used in test cases. Libraries can be built-in (e.g., BuiltIn, Collections), external (e.g., SeleniumLibrary), or custom.

8. How do you import a library in Robot Framework?

Libraries are imported in the `*** Settings ***` section:

```robot
*** Settings ***
Library    SeleniumLibrary
```

9. What are built-in libraries in Robot Framework?

Built-in libraries are included with Robot Framework and provide common functionalities. Examples include BuiltIn, Collections, and String.

10. How do you run a Robot Framework test case?

Use the `robot` command:

```sh
robot path/to/testfile.robot
```

11. What is the difference between a test case and a keyword?

A test case defines a specific scenario to be tested, while a keyword represents an action or a set of actions that can be reused across multiple test cases.

12. How do you define variables in Robot Framework?

Variables are defined in the `*** Variables ***` section:

```robot
*** Variables ***
${USERNAME}    testuser
${PASSWORD}    testpassword
```

13. What are scalar, list, and dictionary variables in Robot Framework?

- **Scalar variables** store a single value (`${VAR}`),
- **list variables** store multiple values (`@{LIST}`), and
- **dictionary variables** store key-value pairs (`&{DICT}`).

14. How do you pass arguments to a user-defined keyword?

Use the `**[Arguments]**` setting in the keyword definition:

```robot
*** Keywords ***
Login To Application
    [Arguments]    ${username}    ${password}
    Input Text     id=username_field    ${username}
    Input Text     id=password_field    ${password}
    Click Button   id=login_button
```

15. How do you handle errors in Robot Framework?

Errors can be handled using `**Run Keyword And Expect Error**`, `**Run Keyword And Ignore Error**`, or using the `**Try Except**` structure available from **Robot Framework 4.0**:

```robot
*** Keywords ***
Handle Error
    [Arguments]    ${keyword}
    Run Keyword And Expect Error    *    ${keyword}
```

16. What is the purpose of the `*** Settings ***` section?

The `*** **Settings** ***` section is used to import libraries, resource files, variable files, and set metadata for the test suite.

17. What is a resource file in Robot Framework?

A resource file is a file containing reusable keywords, variables, and settings that can be imported into test suites and other resource files.

18. How do you import a resource file?

Resource files are imported in the `*** **Settings** ***` section:

```robot
*** Settings ***
Resource    path/to/resourcefile.robot
```

19. What is a test case tag in Robot Framework?

Tags are used to categorise and select test cases for execution. They are specified using the `[Tags]` setting in a test case:

```robot
*** Test Cases ***
Valid Login Test
    [Tags]    smoke    critical
    Open Browser To Login Page  ${LOGIN_URL}
    Login With Valid Credentials  ${USERNAME}  ${PASSWORD}
```

20. How do you run test cases with specific tags?
Use the `-i` (include) or `-e` (exclude) options with the `robot` command:
```sh
robot -i smoke tests/
robot -e regression tests/
```

21-40: Advanced Features and Customization

21. What is the purpose of the `*** Variables ***` section?
- The `*** **Variables** ***` section is used to define scalar, list, and dictionary variables that can be used throughout the test suite.

22. How do you create a list variable in Robot Framework?
List variables are created using the `@{}` syntax:
```robot
*** Variables ***
@{BROWSERS}    chrome    firefox    edge
```

23. How do you create a dictionary variable in Robot Framework?
Dictionary variables are created using the `&{}` syntax:
```robot
*** Variables ***
&{USER}    username=testuser    password=testpassword
```

24. What is the purpose of the `*** Test Cases ***` section?
The `*** **Test Cases** ***` section is where you define individual test cases, including the steps to execute and any settings or tags associated with the test case.

25. How do you execute a specific test case within a test suite?
Use the `**--test**` option with the `robot` command:
```sh
robot --test "Valid Login Test" tests/web_tests/login_tests.robot
```

26. What are the benefits of using keywords in Robot Framework?
Keywords promote reusability, readability, and maintainability of test scripts by encapsulating common actions into reusable units.

27. How do you execute a specific test suite within a directory?

Use the `**--suite**` option with the `robot` command:

```sh
robot --suite suite_name path/to/tests
```

28. What is the purpose of the `*** Documentation ***` section?

The `*** **Documentation** ***` section is used to provide descriptive documentation for the test suite, test case, or keyword.

29. How do you execute tests in parallel using Robot Framework?

Use the Pabot library to execute tests in parallel:

```sh
pip install robotframework-pabot
pabot --processes 4 path/to/tests
```

30. How do you handle setup and teardown in Robot Framework?

Setup and teardown are handled using the `**[Setup]**` and `**[Teardown]**` settings in test cases, keywords, or test suites:

```robot
*** Test Cases ***
Valid Login Test
    [Setup]    Open Browser To Login Page  ${LOGIN_URL}
    [Teardown]    Close Browser
    Login With Valid Credentials  ${USERNAME}  ${PASSWORD}
```

31. How do you create a custom library in Robot Framework?

Create a Python class with methods representing keywords, then import the library in your Robot Framework test suite:

```python
class CustomLibrary:
    def custom_keyword(self, arg1, arg2):
        # Keyword implementation
        pass

if __name__ == '__main__':
    from robot.api.deco import keyword
    CustomLibrary.custom_keyword = keyword(name='Custom Keyword')(CustomLibrary.custom_keyword)
```

```robot
*** Settings ***
Library    path/to/custom_library.py
```

32. How do you use external libraries with Robot Framework?

Install the external library using pip and import it in the `*** **Settings** ***` section:

```sh
pip install robotframework-seleniumlibrary
```

```robot
*** Settings ***
Library    SeleniumLibrary
```

33. How do you pass environment variables to Robot Framework?

Use the `**--variable**` option

with the `robot` command or define them in a variable file:

```sh
robot --variable ENV:PRODUCTION tests/
```

34. What is the purpose of the `*** Metadata ***` section?

The `*** **Metadata** ***` section is used to define metadata for the test suite, such as version information or author details.

35. How do you use the `Run Keyword And Ignore Error` keyword?

Use this keyword to execute another keyword and ignore any errors that occur:

```robot
Run Keyword And Ignore Error    Click Button    id=nonexistent_button
```

36. How do you use the `Run Keyword And Expect Error` keyword?

Use this keyword to execute another keyword and expect a specific error:

```robot
Run Keyword And Expect Error    *    Click Button    id=nonexistent_button
```

37. How do you create a teardown for a test suite?

Use the `Suite Teardown` setting in the `*** **Settings** ***` section:

```robot
*** Settings ***
Suite Teardown    Close All Browsers
```

38. How do you use the `Try Except` structure in Robot Framework?

Use the `Try` and `Except` keywords to handle exceptions:

```robot
Try
    Open Browser    ${LOGIN_URL}    chrome
Except
    Log    Failed to open browser
End
```

39. What is the purpose of the `*** Comments ***` section?

The `*** **Comments** ***` section is used to include comments in the test suite. These comments are ignored during test execution.

40. How do you create a for loop in Robot Framework?

Use the `FOR` keyword to create a loop:

```robot
*** Test Cases ***
Loop Example
    @{ITEMS}    Create List    1    2    3
    FOR    ${item}    IN    @{ITEMS}
        Log    ${item}
    END
```

41-60: SeleniumLibrary and Web Testing

41. What is SeleniumLibrary in Robot Framework?

SeleniumLibrary is an external library for Robot Framework that provides keywords for web testing, allowing interaction with web elements using the Selenium WebDriver.

42. How do you install SeleniumLibrary?

Install SeleniumLibrary using pip:

```sh
pip install robotframework-seleniumlibrary
```

43. How do you open a browser using SeleniumLibrary?

Use the `Open Browser` keyword:

```robot
Open Browser    ${URL}    chrome
```

44. How do you close a browser using SeleniumLibrary?

Use the `Close Browser` keyword:

```robot
Close Browser
```

45. How do you maximise a browser window using SeleniumLibrary?

Use the `Maximise Browser Window` keyword:

```robot
Maximise Browser Window
```

46. How do you navigate to a URL using SeleniumLibrary?

Use the `Go To` keyword:

```robot
Go To    ${URL}
```

47. How do you find an element by its ID using SeleniumLibrary?

Use the `Find Element` keyword with the `id` locator strategy:

```robot
Find Element    id=username_field
```

48. How do you input text into a text field using SeleniumLibrary?

Use the `Input Text` keyword:

```robot
Input Text    id=username_field    ${USERNAME}
```

49. How do you click a button using SeleniumLibrary?

Use the `Click Button` keyword:

```robot
Click Button    id=login_button
```

50. How do you select an option from a dropdown using SeleniumLibrary?

Use the `Select From List By Label` keyword:

```robot
Select From List By Label    id=dropdown    Option 1
```

51. How do you wait for an element to be visible using SeleniumLibrary?

Use the `Wait Until Element Is Visible` keyword:

```robot
Wait Until Element Is Visible    id=logout_button
```

52. How do you wait for an element to be clickable using SeleniumLibrary?

Use the `Wait Until Element Is Clickable` keyword:

```robot
Wait Until Element Is Clickable    id=login_button
```

53. How do you check if an element is visible using SeleniumLibrary?

Use the `Element Should Be Visible` keyword:

```robot
Element Should Be Visible    id=logout_button
```

54. How do you check if an element is not visible using SeleniumLibrary?

Use the `Element Should Not Be Visible` keyword:

```robot
Element Should Not Be Visible    id=logout_button
```

55. How do you get the text of an element using SeleniumLibrary?

Use the `Get Text` keyword:

```robot
${text}    Get Text    id=welcome_message
```

56. How do you verify the title of a page using SeleniumLibrary?

Use the `Title Should Be` keyword:

```robot
Title Should Be    Welcome Page
```

57. How do you capture a screenshot using SeleniumLibrary?

Use the `Capture Page Screenshot` keyword:

```robot
Capture Page Screenshot
```

58. How do you handle alerts using SeleniumLibrary?

Use the `Handle Alert` keyword:

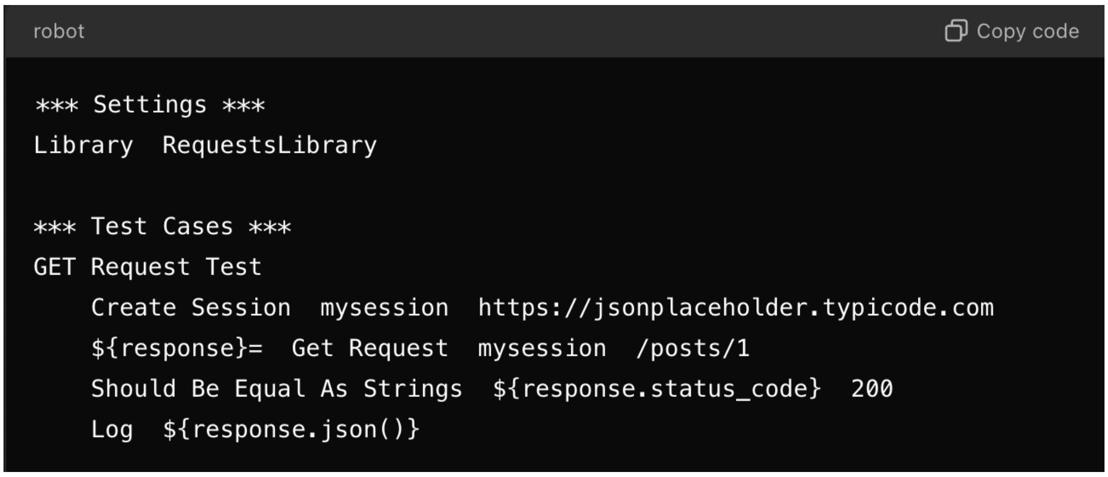
```robot
Handle Alert    ACCEPT
```

59. How do you switch to a frame using SeleniumLibrary?
Use the `Select Frame` keyword:
```robot
Select Frame    id=frame_id
```

60. How do you switch back to the main content from a frame using SeleniumLibrary?
Use the `Unselect Frame` keyword:
```robot
Unselect Frame
```

61-80: API Testing and HTTP Requests

61. How do you perform API testing with Robot Framework?
Use the `RequestsLibrary` or `RESTInstance` library to perform HTTP requests and validate responses.



```robot
*** Settings ***
Library  RequestsLibrary

*** Test Cases ***
GET Request Test
    Create Session  mysession  https://jsonplaceholder.typicode.com
    ${response}=  Get Request  mysession  /posts/1
    Should Be Equal As Strings  ${response.status_code}  200
    Log  ${response.json()}
```

62. How do you install RequestsLibrary for Robot Framework?
Install RequestsLibrary using pip:
```sh
pip install robotframework-requests
```

63. How do you import RequestsLibrary in a test suite?
Import the library in the `*** Settings ***` section:
```robot
*** Settings ***
Library    RequestsLibrary
```

64. How do you create a session in RequestsLibrary?

Use the `Create Session` keyword:
```robot
Create Session    my_session    ${BASE_URL}
```

65. How do you send a GET request using RequestsLibrary?

Use the `Get Request` keyword:
```robot
${response}    Get Request    my_session    /api/resource
```

66. How do you send a POST request using RequestsLibrary?

Use the `Post Request` keyword:
```robot
${response}    Post Request    my_session    /api/resource    ${data}
```

67. How do you validate the status code of a response in RequestsLibrary?

Use the `Should Be Equal As Numbers` keyword:
```robot
${status}    Get Response Status    ${response}
Should Be Equal As Numbers    ${status}    200
```

68. How do you validate the JSON response body in RequestsLibrary?

Use the `Get Response Json` keyword and appropriate assertions:
```robot
${json}    Get Response Json    ${response}
Should Be Equal    ${json['key']}    expected_value
```

69. How do you send a PUT request using RequestsLibrary?

Use the `Put Request` keyword:
```robot
${response}    Put Request    my_session    /api/resource    ${data}
```

70. How do you send a DELETE request using RequestsLibrary?

Use the `Delete Request` keyword:
```robot
${response}    Delete Request    my_session    /api/resource
```

71. How do you add headers to a request in RequestsLibrary?
Use the `Create Session` keyword with the `headers` argument:
```robot
${headers}    Create Dictionary    Authorization=Bearer ${TOKEN}
Create Session    my_session    ${BASE_URL}    headers=${headers}
```

72. How do you add query parameters to a request in RequestsLibrary?
Include the query parameters in the URL or use the `params` argument:
```robot
${params}    Create Dictionary    key=value
${response}    Get Request    my_session    /api/resource    params=${params}
```

73. How do you handle cookies in RequestsLibrary?
Use the `Create Session` keyword with the `cookies` argument:
```robot
${cookies}    Create Dictionary    sessionid=12345
Create Session    my_session    ${BASE_URL}    cookies=${cookies}
```

74. How do you handle basic authentication in RequestsLibrary?
Use the `Create Session` keyword with the `auth` argument:
```robot
Create Session    my_session    ${BASE_URL}
auth=${USERNAME}:${PASSWORD}
```

75. How do you validate the response headers in RequestsLibrary?
Use the `Get Response Headers` keyword and appropriate assertions:
```robot
${headers}    Get Response Headers    ${response}
Should Be Equal    ${headers['Content-Type']}    application/json
```

76. How do you send a multipart/form-data request using RequestsLibrary?**
Use the `Post Request` keyword with the `files` argument:
```robot
${files}    Create Dictionary    file=@path/to/file
${response}    Post Request    my_session    /api/upload    files=${files}
```

77. How do you handle redirection in RequestsLibrary?
Use the `Create Session` keyword with the `allow_redirects` argument:
```robot
Create Session    my_session    ${BASE_URL}    allow_redirects=${False}
```

78. How do you handle timeouts in RequestsLibrary?
Use the `Create Session` keyword with the `timeout` argument:

```robot
Create Session    my_session    ${BASE_URL}    timeout=${30}
```

79. How do you handle SSL verification in RequestsLibrary?
Use the `Create Session` keyword with the `verify` argument:

```robot
Create Session    my_session    ${BASE_URL}    verify=${False}
```

80. How do you validate the response time in RequestsLibrary?
Use the `Get Response Elapsed Time` keyword and appropriate assertions:

```robot
${time}    Get Response Elapsed Time    ${response}
Should Be Less Than    ${time}    ${500}
```

81-100: Database Testing

81. How do you perform database testing with Robot Framework?
Use the `**DatabaseLibrary**` or `**DBLibrary**` to interact with databases and validate results.

82. How do you install DatabaseLibrary for Robot Framework?
Install DatabaseLibrary using pip:

```sh
pip install robotframework-databaselibrary
```

83. How do you import DatabaseLibrary in a test suite?
Import the library in the `*** Settings ***` section:

```robot
*** Settings ***
Library    DatabaseLibrary
```

84. How do you connect to a database using DatabaseLibrary?
Use the `Connect To Database` keyword:

```robot
Connect To Database    pymysql    ${DB_NAME}    ${DB_USER}
${DB_PASSWORD}    ${DB_HOST}
```

85. How do you execute a SQL query using DatabaseLibrary?

Use the `Execute Sql` keyword:

```robot
${result}    Execute Sql    SELECT * FROM table_name
```

86. How do you validate the result of a SQL query in DatabaseLibrary?

Use the `Query` keyword and appropriate assertions:

```robot
${rows}    Query    SELECT * FROM table_name WHERE id=1
Should Be Equal As Numbers    ${len(${rows})}    1
```

87. How do you insert data into a database using DatabaseLibrary?

Use the `Execute Sql` keyword with an INSERT statement:

```robot
Execute Sql    INSERT INTO table_name (column1, column2) VALUES ('value1', 'value2')
```

88. How do you update data in a database using DatabaseLibrary?

Use the `Execute Sql` keyword with an UPDATE statement:

```robot
Execute Sql    UPDATE table_name SET column1='new_value' WHERE id=1
```

89. How do you delete data from a database using DatabaseLibrary?

Use the `Execute Sql` keyword with a DELETE statement:

```robot
Execute Sql    DELETE FROM table_name WHERE id=1
```

90. How do you close the database connection in DatabaseLibrary?

Use the `Disconnect From Database` keyword:

```robot
Disconnect From Database
```

91. How do you handle database transactions in DatabaseLibrary?

Use the `Execute Sql Script` keyword for complex transactions:

```robot
Execute Sql Script
...    BEGIN;
...    INSERT INTO table_name (column1) VALUES ('value1');
...    INSERT INTO table_name (column1) VALUES ('value2');
...    COMMIT;
```

92. How do you handle stored procedures in DatabaseLibrary?
Use the `Call Stored Procedure` keyword:
```robot
Call Stored Procedure    procedure_name    param1    param2
```

93. How do you fetch all rows from a SQL query result in DatabaseLibrary?
Use the `Query` keyword and iterate over the results:
```robot
${rows}    Query    SELECT * FROM table_name
FOR    ${row}    IN    @{rows}
    Log    ${row}
END
```

94. How do you fetch a single row from a SQL query result in DatabaseLibrary?
Use the `Query` keyword and access the first row:
```robot
${rows}    Query    SELECT * FROM table_name WHERE id=1
${row}    Set Variable    ${rows[0]}
```

95. How do you handle database connection pooling in DatabaseLibrary?
DatabaseLibrary does not support connection pooling directly. You need to manage connections explicitly.

96. How do you validate the number of rows returned by a SQL query in DatabaseLibrary?
Use the `Query` keyword and validate the length of the result:
```robot
${rows}    Query    SELECT * FROM table_name
Should Be Equal As Numbers    ${len(${rows})}    10
```

97. How do you handle SQL injection testing in DatabaseLibrary?
Execute potentially malicious SQL inputs and validate that they do not compromise the database:
```robot
${result}    Execute Sql    SELECT * FROM users WHERE username='admin'--'
Should Be Empty    ${result}
```

98. How do you handle different database dialects in DatabaseLibrary?

Use the appropriate database driver (e.g., `pymysql` for MySQL, `psycopg2` for PostgreSQL) when connecting to the database:

```robot
Connect To Database    psycopg2    ${DB_NAME}    ${DB_USER}    ${DB_PASSWORD}    ${DB_HOST}
```

99. How do you log database queries and results in DatabaseLibrary?

Use the `Log` keyword to log queries and results:

```robot
${rows}    Query    SELECT * FROM table_name
Log    ${rows}
```

100. How do you handle database errors in DatabaseLibrary?

Use the `Try Except` structure to handle errors:

```robot
Try
   Execute Sql    INVALID SQL QUERY
Except
   Log    Database query failed
End
```

101-120: Data-Driven Testing

101. What is data-driven testing in Robot Framework?

Data-driven testing involves executing the same test case with different sets of input data to ensure the application behaves as expected with various inputs.

102. How do you implement data-driven testing in Robot Framework?

Use the `FOR` loop or a combination of test templates and external data files.

103. How do you use the `FOR` loop for data-driven testing in Robot Framework?

- Use the `FOR` loop to iterate over a list of data sets:

```robot
*** Test Cases ***
Data Driven Test
   @{DATA}    Create List    data1    data2    data3
   FOR    ${item}    IN    @{DATA}
      Log    ${item}
   END
```

104. How do you create a test template in Robot Framework?

Use the `[Template]` setting to define a test template:

```robot
*** Test Cases ***
Data Driven Test
    [Template]    Template Keyword
    data1    value1
    data2    value2

*** Keywords ***
Template Keyword
    [Arguments]    ${arg1}    ${arg2}
    Log    ${arg1}
    Log    ${arg2}
```

105. How do you use an external CSV file for data-driven testing in Robot Framework?

Use the `DataDriver` library to read data from a CSV file and execute tests for each row:

```robot
*** Settings ***
Library    DataDriver    file=path/to/data.csv

*** Test Cases ***
Data Driven Test
    [Template]    Template Keyword
    @{row}

*** Keywords ***
Template Keyword
    [Arguments]    ${arg1}    ${arg2}
    Log    ${arg1}
    Log    ${arg2}
```

106. How do you use an external Excel file for data-driven testing in Robot Framework?

Use the `**RoboFramework-ExcelDataDriver**` library to read data from an Excel file and execute tests for each row:

```robot
*** Settings ***
Library    ExcelDataDriver    file=path/to/data.xlsx

*** Test Cases ***
Data Driven Test
    [Template]    Template Keyword
    @{row}

*** Keywords ***
Template Keyword
    [Arguments]    ${arg1}    ${arg2}
    Log    ${arg1}
    Log    ${arg2}
```

107. How do you parameterize test cases in Robot Framework?

Use the `Variables` section to define parameters and use them in test cases:

```robot
*** Variables ***
${USERNAME}    test_user
${PASSWORD}    password123

*** Test Cases ***
Login Test
    Input Text    id=username    ${USERNAME}
    Input Text    id=password    ${PASSWORD}
    Click Button    id=login_button
```

108. How do you use the `Variables` section for data-driven testing?

Define variables in the `Variables` section and use them in test cases:

```robot
*** Variables ***
@{USERNAMES}    user1    user2    user3
@{PASSWORDS}    pass1    pass2    pass3

*** Test Cases ***
Data Driven Test
    FOR    ${username}    IN    @{USERNAMES}
        Log    ${username}
    END
```

\`\`\`

109. How do you use the `Variable` keyword to set variables at runtime?
Use the `Set Test Variable` or `Set Suite Variable` keywords to set variables during test execution:
```robot
Set Test Variable    ${USERNAME}    dynamic_user
```

110. How do you use the `Variables` section to import variables from a file?
Use the `Variables` setting to import variables from a file:
```robot
*** Settings ***
Variables    variables.py
```

111. How do you use the `Scalar` variable type in Robot Framework?
Use the `${}` notation for scalar variables:
```robot
${USERNAME}    test_user
```

112. How do you use the `List` variable type in Robot Framework?
Use the `@{}` notation for list variables:
```robot
@{USERNAMES}    user1    user2    user3
```

113. How do you use the `Dictionary` variable type in Robot Framework?
Use the `&{}` notation for dictionary variables:
```robot
&{USER_CREDENTIALS}    username=test_user    password=password123
```

114. How do you use the `Variable` keyword to get the value of a variable?
Use the `Get Variable` keyword to retrieve the value of a variable:
```robot
${value}    Get Variable    ${USERNAME}
```
2

115. How do you use the `Set Variable` keyword to set a variable?
Use the `Set Variable` keyword to set a variable:
```robot
${USERNAME}    Set Variable    dynamic_user
```

116. How do you use the `Set Suite Variable` keyword to set a variable globally?
Use the `Set Suite Variable` keyword to set a variable for the entire test suite:
```robot
Set Suite Variable    ${USERNAME}    suite_user
```

117. How do you use the `Set Test Variable` keyword to set a variable for a test case?
Use the `Set Test Variable` keyword to set a variable for a specific test case:
```robot
Set Test Variable    ${USERNAME}    test_user
```

118. How do you use the `Set Global Variable` keyword to set a variable globally?
Use the `Set Global Variable` keyword to set a variable globally:
```robot
Set Global Variable    ${USERNAME}    global_user
```

119. How do you use the `Create List` keyword to create a list variable?
Use the `Create List` keyword to create a list variable:
```robot
@{USERNAMES}    Create List    user1    user2    user3
```

120. How do you use the `Create Dictionary` keyword to create a dictionary variable?
Use the `Create Dictionary` keyword to create a dictionary variable:
```robot
&{USER_CREDENTIALS}    Create Dictionary    username=test_user
password=password123
```

121-140: Custom Libraries and Keywords

121. How do you create a custom library in Python for Robot Framework?

Create a Python file with functions representing keywords and import it in the Robot Framework test suite:

```python
# custom_library.py
def custom_keyword(arg1, arg2):
    return f"Received {arg1} and {arg2}"
```

```robot
*** Settings ***
Library    custom_library.py

*** Test Cases ***
Use Custom Keyword
    ${result}    custom_keyword    arg1    arg2
    Log    ${result}
```

122. How do you create a custom keyword in Robot Framework?

Use the `*** Keywords ***` section to define a custom keyword:

```robot
*** Keywords ***
Custom Keyword
    [Arguments]    ${arg1}    ${arg2}
    Log    ${arg1}
    Log    ${arg2}
```

123. How do you pass arguments to a custom keyword in Robot Framework?

Use the `[Arguments]` setting to define the arguments for the custom keyword:

```robot
*** Keywords ***
Custom Keyword
    [Arguments]    ${arg1}    ${arg2}
    Log    ${arg1}
    Log    ${arg2}
```

124. How do you return a value from a custom keyword in Robot Framework?
Use the `Return From Keyword` keyword to return a value:

```robot
*** Keywords ***
Custom Keyword
    [Arguments]    ${arg1}    ${arg2}
    ${result}    Set Variable    ${arg1} and ${arg2}
    Return From Keyword    ${result}
```

125. How do you create a custom keyword that calls other keywords in Robot Framework?
Use the custom keyword to call other keywords:

```robot
*** Keywords ***
Custom Keyword
    [Arguments]    ${arg1}    ${arg2}
    Log    ${arg1}
    Log    ${arg2}
    Another Keyword    ${arg1}    ${arg2}

Another Keyword
    [Arguments]    ${arg1}    ${arg2}
    Log    Another keyword received ${arg1} and ${arg2}
```

126. How do you create a custom library in Java for Robot Framework?
Implement the library in Java and use the `Robot Framework Java Library` to expose it to Robot Framework.

127. How do you import a custom Java library in Robot Framework?
Use the `Library` setting to import the custom Java library:

```robot
*** Settings ***
Library    com.example.CustomJavaLibrary
```

128. How do you implement a custom keyword in Java for Robot Framework?
Implement the keyword as a public method in a Java class:

```java
public class CustomJavaLibrary {
    public String customKeyword(String arg1, String arg2) {
        return "Received " + arg1 + " and " + arg2;
    }
}
```

129. How do you return a value from a custom keyword in Java for Robot Framework?

Return the value from the Java method:

```java
public class CustomJavaLibrary {
    public String customKeyword(String arg1, String arg2) {
        return "Received " + arg1 + " and " + arg2;
    }
}
```

130. How do you handle exceptions in custom keywords in Python for Robot Framework?

Use try-except blocks in the Python function:

```python
def custom_keyword(arg1, arg2):
    try:
        # Your code here
        return f"Received {arg1} and {arg2}"
    except Exception as e:
        return f"Error: {str(e)}"
```

131. How do you handle exceptions in custom keywords in Java for Robot Framework?

Use try-catch blocks in the Java method:

```java
public class CustomJavaLibrary {
    public String customKeyword(String arg1, String arg2) {
        try {
            return "Received " + arg1 + " and " + arg2;
        } catch (Exception e) {
            return "Error: " + e.getMessage();
        }
    }
}
```

132. How do you create a custom library in JavaScript for Robot Framework?

Implement the library in JavaScript and use a JavaScript library integration tool like `RobotJS`.

133. How do you import a custom JavaScript library in Robot Framework?

Use the `Library` setting to import the custom JavaScript library:

```robot
*** Settings ***
Library    custom_javascript_library.js
```

~~~
```
~~~

134. How do you implement a custom keyword in JavaScript for Robot Framework?

Implement the keyword as a function in a JavaScript file:

```javascript
function customKeyword(arg1, arg2) {
  return `Received ${arg1} and ${arg2}`;
}
```

135. How do you return a value from a custom keyword in JavaScript for Robot Framework?

Return the value from the JavaScript function:

```javascript
function customKeyword(arg1, arg2) {
  return `Received ${arg1} and ${arg2}`;
}
```

136. How do you handle exceptions in custom keywords in JavaScript for Robot Framework?

Use try-catch blocks in the JavaScript function:

```javascript
function customKeyword(arg1, arg2) {
  try {
    return `Received ${arg1} and ${arg2}`;
  } catch (e) {
    return `Error: ${e.message}`;
  }
}
```

137. How do you create a custom library in C# for Robot Framework?

Implement the library in C# and use a .NET integration tool like `RobotDotNet`.

138. How do you import a custom C# library in Robot Framework?

Use the `Library` setting to import the custom C# library:

```robot
*** Settings ***
Library    CustomCSharpLibrary
```

139. How do you implement a custom keyword in C# for Robot Framework?

Implement the keyword as a method in a C# class:

```csharp
public class CustomCSharpLibrary {
   public string CustomKeyword(string arg1, string arg2) {
      return $"Received {arg1} and {arg2}";
   }
}
```

140. How do you return a value from a custom keyword in C# for Robot Framework?

Return the value from the C# method:

```csharp
public class CustomCSharpLibrary {
   public string CustomKeyword(string arg1, string arg2) {
      return $"Received {arg1} and {arg2}";
   }
}
```

141-160: Advanced Usage

141. How do you handle conditional execution in Robot Framework?

Use the `Run Keyword If` keyword for conditional execution:

```robot
Run Keyword If    '${condition}' == 'value'    Keyword To Run
```

142. How do you handle loops in Robot Framework?

Use the `FOR` loop for iterating over a list:

```robot
FOR    ${item}    IN    @{LIST}
   Log    ${item}
END
```

143. How do you run keywords conditionally based on the outcome of previous keywords?

Use the `Run Keyword If` keyword with a conditional expression:

```robot
Run Keyword If    '${status}' == 'PASS'    Keyword To Run
```

144. How do you skip a test case in Robot Framework?

Use the `Skip` keyword to skip a test case:

```robot
Skip    Skipping this test case
```

145. How do you run a keyword with a timeout in Robot Framework?

Use the `Run Keyword With Timeout` keyword to specify a timeout for the keyword:

```robot
Run Keyword With Timeout    10s    Keyword To Run
```

146. How do you run a keyword in a separate thread in Robot Framework?

Use the `Run Keyword In Separate Thread` keyword to run the keyword in a separate thread:

```robot
Run Keyword In Separate Thread    Keyword To Run
```

147. How do you wait for a keyword to complete in Robot Framework?

Use the `Wait Until Keyword Succeeds` keyword to wait for the keyword to complete:

```robot
Wait Until Keyword Succeeds    10s    1s    Keyword To Wait For
```

148. How do you run a keyword and ignore its failure in Robot Framework?

Use the `Run Keyword And Ignore Error` keyword to run the keyword and ignore its failure:

```robot
Run Keyword And Ignore Error    Keyword That Might Fail
```

149. How do you run multiple keywords sequentially in Robot Framework?

Use the `Run Keywords` keyword to run multiple keywords sequentially:

```robot
Run Keywords    Keyword 1    AND    Keyword 2    AND    Keyword 3
```

150. How do you handle keyword execution failures in Robot Framework?

Use the `Run Keyword And Return Status` keyword to handle execution failures:

```robot
${status}    Run Keyword And Return Status    Keyword That Might Fail
```

151. How do you run a keyword and return its result in Robot Framework?

- Use the `Run Keyword And Return` keyword to run the keyword and return its result:

```robot
${result}    Run Keyword And Return    Keyword To Run
```

152. How do you run a keyword and store its result in a variable in Robot Framework?

Use the `Run Keyword And Return` keyword to run the keyword and store its result in a variable:

```robot
${result}    Run Keyword And Return    Keyword To Run
```

153. How do you run a keyword and return its status and output in Robot Framework?

Use the `Run Keyword And Return Status And Output` keyword to run the keyword and return its status and output:

```robot
${status}    ${output}    Run Keyword And Return Status And Output    Keyword To Run
```

154. How do you run a keyword conditionally based on multiple conditions in Robot Framework?

Use the `Run Keyword If` keyword with multiple conditions:

```robot
Run Keyword If    '${condition1}' == 'value1' and '${condition2}' == 'value2'    Keyword To Run
```

155. How do you run a keyword for each item in a list in Robot Framework?

Use the `FOR` loop to run the keyword for each item in a list:

```robot
FOR    ${item}    IN    @{LIST}
    Keyword To Run    ${item}
END
```

156. How do you run a keyword for each item in a list with a specific index in Robot Framework?

Use the `FOR` loop with index to run the keyword for each item in a list:

```robot
FOR    ${index}    ${item}    IN ENUMERATE    @{LIST}
    Keyword To Run    ${index}    ${item}
END
```

157. How do you run a keyword with arguments in Robot Framework?

Use the `Run Keyword` keyword with arguments:

```robot
Run Keyword    Keyword To Run    arg1    arg2
```

158. How do you run a keyword with arguments and store the result in Robot Framework?

Use the `Run Keyword And Return` keyword with arguments and store the result:

```robot
${result}    Run Keyword And Return    Keyword To Run    arg1    arg2
```

159. How do you run a keyword and log its output in Robot Framework?

Use the `Run Keyword And Log` keyword to run the keyword and log its output:

```robot
Run Keyword And Log    Keyword To Run
```

160. How do you run a keyword and log its status and output in Robot Framework?

Use the `Run Keyword And Log Status And Output` keyword to run the keyword and log its status and output:

```robot
Run Keyword And Log Status And Output    Keyword To Run
```

161. What are the key features of Robot Framework?

- Keyword-driven and data-driven testing
- Tabular test data syntax
- Clear reports and logs in HTML
- Extensible with libraries written in Python or Java
- Supports both web and non-web testing

162. What is a keyword in Robot Framework?

- A keyword is a single action that is used to perform a specific task in a test case. Keywords can be user-defined or come from standard or external libraries.

163. What are the built-in libraries in Robot Framework?
- BuiltIn
- Collections
- DateTime
- Dialogs
- OperatingSystem
- Process
- Screenshot
- String
- Telnet
- XML

164. How do you create a user-defined keyword in Robot Framework?
- User-defined keywords are created in the *** Keywords *** section of a .robot file or in resource files.

```robot
*** Keywords ***
Login
    [Arguments]    ${username}    ${password}
    Input Text    id=username_field    ${username}
    Input Text    id=password_field    ${password}
    Click Button  id=login_button
```

165. How do you run a Robot Framework test case?
- Use the `robot` command followed by the path to the test case file or directory.

```sh
robot path/to/testfile.robot
```

166. What are test suites in Robot Framework?
- A test suite is a collection of test cases, often organised in directories and subdirectories. Test suites can also include other test suites, forming a hierarchy.

167. How can you include or exclude tests based on tags?
- Use the `-i` option to include and `-e` option to exclude tests based on tags.

```sh
robot -i smoke path/to/tests
robot -e regression path/to/tests
```

168. How do you handle variables in Robot Framework?

- Variables can be defined in the *** Variables *** section, or passed from command line or variable files.

```robot
*** Variables ***
${URL}    http://example.com
${USERNAME}  testuser
${PASSWORD}  testpassword
```

169. Explain the difference between resource files and library files.

- Resource files contain user-defined keywords, variables, and settings, and are written in Robot Framework syntax. They are imported using the `Resource` setting.
- Library files are Python or Java files that provide additional functionality. They are imported using the `Library` setting.

170. What is the difference between `Run Keyword If` and `Run Keywords`?

- `Run Keyword If` executes a keyword conditionally based on the given condition.

```robot
Run Keyword If  ${condition}  Some Keyword
```

- `Run Keywords` allows you to run multiple keywords in a sequence.

```robot
Run Keywords  Keyword1  AND  Keyword2
```

171. How do you handle exceptions in Robot Framework?

- Use the `Run Keyword And Ignore Error` keyword to run a keyword and ignore its possible failure.

```robot
Run Keyword And Ignore Error  Some Keyword
```

172. How do you use the RESTinstance library for API testing in Robot Framework?

- Install the RESTinstance library using pip and import it in your test cases.

```sh
pip install RESTinstance
```

```robot
*** Settings ***
Library  RESTinstance
```

Define and call RESTful service requests using keywords like `GET`, `POST`, `PUT`, etc.

173. Explain how you can use data-driven testing in Robot Framework.

- Use the `FOR` loop and/or variable files to iterate over test data.

```robot
*** Test Cases ***
Data Driven Test
    [Template]  Test Template
    value1  value2
    value3  value4
```

174. How do you generate HTML reports in Robot Framework?

- Robot Framework automatically generates HTML reports and logs in the specified output directory when you run your tests using the `robot` command.

175. What is a resource file and how do you use it?

- A resource file contains reusable keywords, variables, and settings, and is imported using the `Resource` setting.

```robot
*** Settings ***
Resource  path/to/resource_file.robot
```

176. What is the difference between `Run Keyword And Continue On Failure` and `Run Keyword And Ignore Error`?

- `Run Keyword And Continue On Failure` runs the keyword and continues execution regardless of failure, but logs the failure.

```robot
Run Keyword And Continue On Failure  Some Keyword
```

- **`Run Keyword And Ignore Error` runs the keyword and ignores any failure without logging it.**

```robot
Run Keyword And Ignore Error  Some Keyword
```

177. How do you handle dynamic content in Robot Framework?

- Use keywords like `Wait Until Page Contains Element` or `Wait Until Element Is Visible` to handle dynamic content.

```robot
Wait Until Element Is Visible  xpath=//button[@id='submit']
```

178. How do you run tests in parallel using Robot Framework?
- Use the `pabot` tool to run tests in parallel.

bash Copy code

```bash
pabot --processes 4 my_test_suite.robot
```

179. How do you manage dependencies in Robot Framework?
- Use the `requirements.txt` file to manage dependencies and install them using pip.

```sh
pip install -r requirements.txt
```

180. Explain the architecture of Robot Framework.
- Robot Framework has a modular architecture, comprising a core framework that provides the test execution environment, test libraries that extend functionality, and user-defined keywords. The core framework handles reading test data, running test cases, and generating reports.

181. What are fixtures in Robot Framework?

Fixtures in Robot Framework are setup and teardown keywords that are executed before and after test cases, suites, or entire test runs to set up preconditions and clean up afterward.

182. How do you integrate Robot Framework with CI/CD tools?
- You can integrate Robot Framework with CI/CD tools like Jenkins, GitLab CI/CD, or GitHub Actions by creating pipeline scripts that install dependencies and execute Robot Framework tests.

```groovy
pipeline {
    agent any
    stages {
        stage('Install Dependencies') {
            steps {
                sh 'pip install -r requirements.txt'
            }
        }
        stage('Run Tests') {
            steps {
                sh 'robot -d reports tests/'
            }
        }
    }
    post {
        always {
            archiveArtifacts artifacts: 'reports/*', allowEmptyArchive: true
            junit 'reports/*.xml'
            }
        }
    }
```

183. What is RPA and how does Robot Framework support it?
   - RPA (Robotic Process Automation) is the use of software robots to automate repetitive tasks. Robot Framework supports RPA through libraries like RPA Framework, which provides keywords for interacting with desktop applications, web applications, and more.

184. How do you implement BDD with Robot Framework?
   - Robot Framework supports BDD (Behavior-Driven Development) by allowing test cases to be written in a behavior-driven style using keywords that reflect user stories or behaviors.

```robot
*** Test Cases ***
User Can Log In
   Given User is on login page
   When User enters valid credentials
   Then User is redirected to homepage
```

185. What are some best practices for writing test cases in Robot Framework?
   - Use descriptive names for test cases and keywords.
   - Reuse keywords to avoid duplication.
   - Keep test cases and keywords simple and focused.
   - Use resource files for shared keywords and variables.
   - Organise test cases in a logical and maintainable directory structure.

186. How do you handle data-driven tests with external data sources in Robot Framework?
- Use the `DataDriver` library to read data from external sources like CSV, Excel, or databases and run data-driven tests.

```robot
*** Settings ***
Library  DataDriver
```

187. What is the purpose of the `robot.yaml` file?
   - The **`robot.yaml`** file is used for configuration when running tests with the `robot` command. It can specify test settings, variables, and other options.

188. How do you extend Robot Framework with custom libraries?

- Write custom Python or Java libraries and import them in your test cases. Use decorators like `robot.api.deco.keyword` in Python to define custom keywords.

```python
from robot.api.deco import keyword
class CustomLibrary:
  @keyword
  def custom_keyword(self, arg1, arg2):
    pass
```

189. How do you handle asynchronous operations in Robot Framework?

- Handle asynchronous operations by using keywords that wait for elements or conditions, such as **`Wait Until Element Is Visible`** or **`Wait Until Keyword Succeeds`**.

190. How do you open a browser in Robot Framework using SeleniumLibrary?

- Use the `Open Browser` keyword from the SeleniumLibrary.

```robot
*** Settings ***
Library  SeleniumLibrary

*** Test Cases ***
Open Browser Test
  Open Browser  http://example.com  chrome
```

191. How do you locate elements using SeleniumLibrary?

- Use keywords like `Click Element`, `Input Text`, `Wait Until Element Is Visible`, etc., and specify locators using strategies like id, name, xpath, css selector, etc.

```robot
Click Element  id=submit_button
```

192. What are implicit and explicit waits in SeleniumLibrary?

- Implicit waits set a default wait time for finding elements, applicable to all element searches.

```robot
Set Selenium Implicit Wait  10 seconds
```

- **Explicit waits wait for specific conditions to be met before proceeding.**

```robot
Wait Until Element Is Visible  id=submit_button
```

193. How do you handle dropdowns in SeleniumLibrary?
  - Use the `Select From List By Label`, `Select From List By Value`, or `Select From List By Index` keywords.

```robot
Select From List By Label  id=dropdown  option1
```

194. How do you handle alerts and pop-ups in SeleniumLibrary?
  - Use keywords like `Handle Alert`, `Dismiss Alert`, and `Alert Should Be Present`.

```robot
Handle Alert  OK
```

195. How do you upload a file using SeleniumLibrary?
  - Use the `Choose File` keyword to upload a file.

```robot
Choose File  id=file_input  path/to/file
```

196. How do you handle frames and iframes in SeleniumLibrary?
  - Use the `Select Frame` and `Unselect Frame` keywords to switch to and from frames.

```robot
Select Frame  id=frame_id
Unselect Frame
```

197. How do you scroll a page in SeleniumLibrary?
  - Use the **`Execute Javascript`** keyword to scroll the page.

```robot
Execute Javascript  window.scrollTo(0, document.body.scrollHeight);
```

198. How do you capture a screenshot of failure in SeleniumLibrary?
  - Use the `Capture Page Screenshot` keyword in a `Test Teardown` section or as part of an error handling keyword.

```robot
*** Test Cases ***
Test With Screenshot On Failure
   [Teardown]  Capture Page Screenshot
   # Test steps here
```

199. How do you maximise the browser window in SeleniumLibrary?
  - Use the `Maximise Browser Window` keyword.

```robot
Maximise Browser Window
```

200. How do you perform API testing using Robot Framework?

- Use libraries like `RequestsLibrary`, `RESTinstance`, or `HttpLibrary`.

```robot
*** Settings ***
Library  RequestsLibrary

*** Test Cases ***
GET Request Test
   Create Session  api  https://jsonplaceholder.typicode.com
   ${response}=  Get Request  api  /todos/1
   Should Be Equal As Strings  ${response.status_code}  200
```

201. What is the RequestsLibrary in Robot Framework?

- RequestsLibrary is a Robot Framework library for HTTP requests, providing keywords to perform GET, POST, PUT, DELETE, and other HTTP methods.

202. How do you create a session in RequestsLibrary?

- Use the `Create Session` keyword.

```robot
Create Session  api  https://api.example.com
```

203. How do you send a GET request in RequestsLibrary?

- Use the `Get Request` keyword.

```robot
${response}=  Get Request  api  /endpoint
```

204. How do you send a POST request in RequestsLibrary?

- Use the `Post Request` keyword.

```robot
${response}=  Post Request  api  /endpoint  ${payload}
```

205. How do you handle response status codes in RequestsLibrary?

- Use keywords like `Should Be Equal As Strings` to verify status codes.

```robot
Should Be Equal As Strings  ${response.status_code}  200
```

206. How do you handle JSON responses in RequestsLibrary?

- Use keywords like `Convert To Dictionary` to parse JSON responses.

```robot
${json}=  Convert To Dictionary  ${response.content}
```

207. How do you add headers to a request in RequestsLibrary?

- Use the `Create Session` or `Update Session` keywords to add headers.

```robot
Create Session  api  https://api.example.com  headers={'Content-Type': 'application/json'}
```

208. How do you perform authentication in RequestsLibrary?

- Use the `Create Session` keyword with authentication parameters.

```robot
Create Session  api  https://api.example.com  auth=('username', 'password')
```

209. How do you validate response time in API testing with RequestsLibrary?

- Use keywords to measure and validate response time.

```robot
${response}=  Get Request  api  /endpoint
${elapsed_time}=  Get From Dictionary  ${response}  elapsed
Should Be Less Than  ${elapsed_time.total_seconds()}  2
```

210. How do you integrate Robot Framework with Jenkins?

- Use a Jenkins pipeline or freestyle project to install dependencies and run Robot Framework tests.

```groovy
pipeline {
   agent any
   stages {
      stage('Install Dependencies') {
         steps {
            sh 'pip install -r requirements.txt'
         }
      }
      stage('Run Tests') {
         steps {
            sh 'robot -d reports tests/'
         }
      }
   }
   post {
      always {
            archiveArtifacts artifacts: 'reports/*', allowEmptyArchive: true
            junit 'reports/*.xml'
        }
    }
}
```

211. How do you integrate Robot Framework with GitLab CI/CD?
  - Use a `.gitlab-ci.yml` file to define the CI/CD pipeline.

```yaml
stages:
  - test

test:
  stage

: test
  script:
    - pip install -r requirements.txt
    - robot -d reports tests/
  artifacts:
    paths:
      - reports/
```

212. How do you use Docker with Robot Framework?
  - Create a Dockerfile to define the environment and use it to run tests in a container.

```Dockerfile
FROM python:3.9-slim

RUN pip install robotframework robotframework-seleniumlibrary

WORKDIR /tests

COPY . .

ENTRYPOINT ["robot"]
```

213. How do you integrate Robot Framework with BrowserStack or Sauce Labs?
  - Use Selenium remote WebDriver to connect to BrowserStack or Sauce Labs.

```robot
Open Browser  http://example.com  browser=${BROWSER}
remote_url=${REMOTE_URL}

```

214. How do you manage test environments in Robot Framework?

- Use variables and variable files to manage different test environments.

```robot
*** Variables ***
${URL}  http://test-environment.com
```

215. How do you handle cross-browser testing in Robot Framework?

- Use the SeleniumLibrary and pass different browser names to the `Open Browser` keyword.

```robot
Open Browser  http://example.com  chrome
Open Browser  http://example.com  firefox
```

216. How do you schedule Robot Framework tests?

- Use a CI/CD tool like Jenkins or cron jobs to schedule tests.

```sh
0 0 * * * cd /path/to/tests && robot -d reports .
```

217. How do you monitor test results over time?

- Use tools like Allure Report, Jenkins Test Results Analyzer, or custom dashboards to track and analyse test results over time.

218. How do you handle test data in Robot Framework?

- Use variable files, external data sources, or data-driven tests to manage and reuse test data.

219. How do you integrate Robot Framework with a test management tool like TestRail?

- Use APIs provided by the test management tool to integrate with Robot Framework.

```robot
*** Settings ***
Library  TestRailLibrary

*** Test Cases ***
Test Case Example
   Add Test Result To TestRail  1  passed
```

220. What are some best practices for organising Robot Framework test cases?

- Group related test cases in directories.
- Use descriptive names for test cases and keywords.
- Keep test cases short and focused.
- Reuse keywords to avoid duplication.
- Separate test data from test logic using variables and variable files.

221. How do you maintain test scripts in Robot Framework?
- Regularly review and refactor test scripts.
- Use version control systems like Git.
- Write clear and concise keywords.
- Use comments and documentation for clarity.

222. How do you ensure reusability of keywords in Robot Framework?
- Create reusable keywords in resource files.
- Use libraries for common functionalities.
- Modularize test cases by breaking them into smaller, reusable components.

223. What is a test strategy in the context of Robot Framework?
- A test strategy outlines the approach for testing, including test types, tools, environments, schedules, and responsibilities. It ensures a systematic and consistent approach to testing.

224. How do you handle version control for Robot Framework projects?
- Use Git or other version control systems to manage code, track changes, and collaborate with team members.

225. What is the role of a test lead in Robot Framework projects?
- A test lead coordinates testing activities, defines test strategies, ensures quality standards, manages test resources, and communicates with stakeholders.

226. How do you ensure test coverage in Robot Framework?
- Use code coverage tools, define comprehensive test cases, and perform regular reviews to ensure all critical paths and scenarios are covered.

227. How do you handle flaky tests in Robot Framework?
- Identify the root cause of flakiness, improve stability by adding waits or retries, and ensure a stable test environment.

228. How do you write maintainable test scripts in Robot Framework?
- Follow best practices like using meaningful names, reusing keywords, keeping tests modular, and documenting code.

229. What is the Page Object Model (POM) and how do you implement it in Robot Framework?

- POM is a design pattern that helps in creating an object repository for web UI elements. Implement it by creating separate resource files for each page and defining keywords to interact with the elements on that page.

```robot
*** Settings ***
Resource  LoginPage.robot
Resource  HomePage.robot

*** Test Cases ***
Login Test
   Open Browser  ${URL}  chrome
   Login Page.Open Login Page
   Login Page.Enter Credentials  ${USERNAME}  ${PASSWORD}
   Login Page.Submit
   Home Page.Verify User Logged In
```

230. How do you debug Robot Framework test cases?

- Use the `--debug` or `--loglevel TRACE` options to get detailed logs.

```sh
robot --loglevel TRACE tests/
```

231. What are some common issues faced in Robot Framework and their solutions?

- Element not found: Use proper waits or check locators.
- Timeout errors: Increase wait time or optimise scripts.
- Environment issues: Ensure consistent test environments.

232. How do you validate test results in Robot Framework?

- Use assertion keywords like **`Should Be Equal`**, **`Should Contain`**, and **`Should Be True`**.

```robot
Should Be Equal  ${result}  expected_value
```

233. How do you log information in Robot Framework?

- Use the `Log` keyword to log information.

```robot
Log  This is a log message
```

234. How do you capture and handle screenshots in Robot Framework?
- Use the `Capture Page Screenshot` keyword to capture screenshots.
```robot
Capture Page Screenshot
```

235. How do you handle dynamic elements in Robot Framework?
- Use dynamic locators, waits, and conditional statements to handle dynamic elements.

236. How do you handle timeouts in Robot Framework?
- Use the `Set Selenium Timeout` keyword to set timeouts for SeleniumLibrary.
```robot
Set Selenium Timeout  10 seconds
```

237. How do you handle test dependencies in Robot Framework?
- Use setup and teardown keywords to handle dependencies and clean up after tests.

238. How do you troubleshoot test failures in Robot Framework?
- Analyse logs and reports, debug scripts, verify locators, and check for environment issues.

239. What is the Dialogs library in Robot Framework?
- The Dialogs library provides keywords to create interactive dialogs, useful for debugging and getting input during test execution.
```robot
${input}=  Get Value From User  Please enter a value:
```

240. How do you use the Collections library in Robot Framework?
- The Collections library provides keywords for handling lists and dictionaries.
```robot
Append To List  ${list}  item
```

241. What is the OperatingSystem library in Robot Framework?
- The OperatingSystem library provides keywords for interacting with the operating system, such as file and directory operations.
```robot
Create Directory  ${path}
```

242. How do you use the String library in Robot Framework?
- The String library provides keywords for string manipulation.
```robot
${result}=  Replace String  Hello world  world  Robot Framework
```

243. How do you use the DateTime library in Robot Framework?
- The DateTime library provides keywords for handling date and time.
```robot
${date}=  Get Current Date
```

244. What is the Process library in Robot Framework?
- The Process library provides keywords to run processes in the operating system and interact with them.
```robot
Run Process  python  -c  print("Hello, world!")
```

245. How do you use the Telnet library in Robot Framework?
- The Telnet library provides keywords for Telnet connections.
```robot
Open Connection  ${host}
```

246. What is the XML library in Robot Framework?
- The XML library provides keywords for handling XML files and data.
```robot
Parse XML  ${xml_string}
```

247. How do you use the DatabaseLibrary in Robot Framework?
- The DatabaseLibrary provides keywords to interact with databases.
```robot
Connect To Database  pymysql  ${db_name}  ${db_username}  ${db_password}
```

248. How do you use the SSHLibrary in Robot Framework?
- The SSHLibrary provides keywords to interact with SSH servers.
```robot
Open Connection  ${host}
Login  ${username}  ${password}
```

249. How do you optimise Robot Framework test performance?
- Minimise wait times, reuse browser sessions, and optimise locators.

250. How do you use `robot.api` for advanced scripting in Robot Framework?
- Use `robot.api` to access and manipulate test data, logs, and reports programmatically.
```python
from robot.api import ExecutionResult
result = ExecutionResult('output.xml')
print(result.suite.statistics.all.total)
```

251. What is the purpose of the `robot.libdoc` module?
- The `robot.libdoc` module generates documentation for libraries and resource files.
```sh
python -m robot.libdoc mylibrary.py mylibrary.html
```

252. How do you create custom listeners in Robot Framework?
- Implement a listener interface in Python to create custom listeners.
```python
from robot.api import logger
from robot.api.deco import keyword

class MyListener:
    ROBOT_LISTENER_API_VERSION = 2

    def start_suite(self, name, attrs):
        logger.info("Starting suite: %s" % name)
```

253. How do you perform parallel test execution in Robot Framework?
- Use the `Pabot` tool for parallel test execution.
```sh
pabot --processes 4 --outputdir results tests/
```

254. How do you handle conditional logic in Robot Framework?
- Use the `Run Keyword If` and `Run Keyword Unless` keywords for conditional logic.
```robot
Run Keyword If  ${condition}  Keyword
```

255. How do you interact with REST APIs using Robot Framework?
- Use the `RequestsLibrary` or `RESTinstance` for interacting with REST APIs.
```robot
Create Session  api  https://jsonplaceholder.typicode.com
${response}=  Get Request  api  /todos/1
```

256. How do you use regular expressions in Robot Framework?
- Use the `Regexp Escape` and `Should Match Regexp` keywords for regular expression operations.

```robot
${pattern}=  Regexp Escape  \d{3}-\d{2}-\d{4}
Should Match Regexp  123-45-6789  ${pattern}
```

257. How do you implement custom error handling in Robot Framework?
- Use the `Run Keyword And Expect Error` keyword for custom error handling.

```robot
Run Keyword And Expect Error  *Error  Keyword
```

258. How do you create complex test setups and teardowns in Robot Framework?
- Use nested setups and teardowns at the suite and test case levels.

```robot
*** Settings ***
Suite Setup  Suite Setup Keyword
Suite Teardown  Suite Teardown Keyword
```

259. How would you test login functionality using Robot Framework?
- Write a test case that opens the login page, enters credentials, submits the form, and verifies the login.

```robot
*** Test Cases ***
Test Login
   Open Browser  ${URL}  chrome
   Input Text  username  ${USERNAME}
   Input Text  password  ${PASSWORD}
   Click Button  login_button
   Element Should Be Visible  homepage
```

260. How would you test a REST API using Robot Framework?
- Write a test case that sends requests to the API and validates the responses.

```robot
*** Test Cases ***
Test GET Request
   Create Session  api  https://jsonplaceholder.typicode.com
   ${response}=  Get Request  api  /todos/1
   Should Be Equal As Strings  ${response.status_code}  200
```

261. How would you handle file uploads and downloads in Robot Framework?
- Use the `Choose File` keyword for uploads and verify downloads by checking the file system.

```robot
Choose File  id=file_input  path/to/file
```

262. How would you automate a web form submission using Robot Framework?
- Write a test case that fills out the form fields and submits the form.

```robot
*** Test Cases ***
Test Form Submission
   Open Browser  ${URL}  chrome
   Input Text  name  John Doe
   Input Text  email  john.doe@example.com
   Click Button  submit_button
   Element Should Be Visible  success_message
```

263. How would you test a dynamic web table using Robot Framework?
- Use loops and conditional logic to interact with and verify table data.

```robot
*** Test Cases ***
Test Dynamic Table
   Open Browser  ${URL}  chrome
   ${rows}=  Get Element Count  //table[@id='table']/tbody/tr
   :FOR  ${row}  IN RANGE  ${rows}
   \  ${cell}=  Get Text  //table[@id='table']/tbody/tr[${row}]/td[1]
   \  Should Be Equal  ${cell}  Expected Value
```

264. How would you perform a data-driven test using Robot Framework?
- Use the `DataDriver` library to read data from an external source and execute tests for each data set.

```robot
*** Settings ***
Library  DataDriver
```

265. How would you test email functionality using Robot Framework?

- Use the `SMTP` and `IMAP` libraries to send and receive emails, then verify the content.

```robot
*** Settings ***
Library  SMTPLibrary
Library  IMAPLibrary

*** Test Cases ***
Test Email
   Open Connection  smtp.example.com
   Send Message  sender@example.com  recipient@example.com  Subject  Body
   Open Mailbox  imap.example.com  user  password
   List Messages  ALL
```

266. How would you test a mobile application using Robot Framework?

- Use the `AppiumLibrary` to interact with mobile applications.

```robot
*** Settings ***
Library  AppiumLibrary

*** Test Cases ***
Test Mobile App
   Open Application  http://localhost:4723/wd/hub  platformName=Android
deviceName=emulator
   Click Element  //android.widget.Button[@content-desc="Login"]
```

267. How would you test a desktop application using Robot Framework?

- Use the `RPA.Desktop` library to interact with desktop applications.

```robot
*** Settings ***
Library  RPA.Desktop

*** Test Cases ***
Test Desktop App
   Open Application  path/to/application
   Click  coordinate=(100, 200)
```

268. How would you perform localization testing using Robot Framework?
- Use variable files or data sources to test different locales and verify the UI and content in various languages.
```robot
*** Test Cases ***
Test Localization
   [Template]  Verify Localization
   en_US  Welcome
   fr_FR  Bienvenue
```

269. What is the difference between `Run Keyword If` and `Run Keyword And Ignore Error`?
- `Run Keyword If` executes a keyword conditionally, while `Run Keyword And Ignore Error` executes a keyword and ignores any errors that occur.
```robot
Run Keyword If  ${condition}  Keyword
Run Keyword And Ignore Error  Keyword
```

270. How do you handle browser cookies in Robot Framework?
- Use the `Get Cookie` and `Add Cookie` keywords from SeleniumLibrary to manage cookies.
```robot
Get Cookie  cookie_name
```

271. What is the purpose of the `BuiltIn` library in Robot Framework?
- The `BuiltIn` library provides a set of common keywords that are always available, like `Log`, `Sleep`, and `Set Variable`.
```robot
Log  This is a log message
```

272. How do you execute shell commands in Robot Framework?
- Use the `OperatingSystem` library to execute shell commands.
```robot
Run Process  ls  -l
```

273. What is the purpose of `Wait Until Keyword Succeeds`?
- `Wait Until Keyword Succeeds` retries a keyword until it succeeds or the specified timeout is reached.
```robot
Wait Until Keyword Succeeds  1 min  5 sec  Keyword
```

274. How do you handle keyboard and mouse actions in Robot Framework?
- Use the `SeleniumLibrary` or `RPA.Desktop` for keyboard and mouse actions.

```robot
Press Keys  locator  ENTER
Click  coordinate=(100, 200)
```

275. How do you manage test dependencies in Robot Framework?
- Use setup and teardown keywords to handle dependencies and ensure proper test environments.

276. What are resource files in Robot Framework?
- Resource files contain reusable keywords and variables that can be imported into test cases.

```robot
*** Settings ***
Resource  common_resources.robot
```

277. How do you perform accessibility testing using Robot Framework?
- Use libraries like `Pa11yLibrary` to perform accessibility testing.

```robot
Library  Pa11yLibrary

*** Test Cases ***
Test Accessibility
   Run Pa11y  http://example.com
```

278. How do you write a custom library in Python for Robot Framework?
- Create a Python class with methods that you want to implement.

A typical folder structure for a Robot Framework project is designed to promote organisation, maintainability, and reusability of test scripts and resources. Below is an explanation of the folder structure along with a brief description of each component:

Generic Robot Framework Folder Structure

```
robot-framework-project/
│
├── tests/
│   ├── test_suites/
│   │   ├── suite1/
│   │   │   └── test_case1.robot
│   │   │   └── test_case2.robot
│   │   └── suite2/
│   │       └── test_case1.robot
│   │
│   │       └── test_case2.robot
│   └── test_cases/
│       └── test_case1.robot
│       └── test_case2.robot
│
├── resources/
│   ├── keywords/
│   │   └── custom_keywords.robot
│   ├── variables/
│   │   └── common_variables.robot
│   └── locators/
│       └── page_locators.robot
│
├── libraries/
│   └── custom_library.py
│
├── results/
│   ├── log.html
│   ├── report.html
│   └── output.xml
│
├── configs/
│   └── config1.py
│   └── config2.py
│
├── logs/
│   ├── debug.log
│   └── errors.log
│
├── data/
│   └── test_data.xlsx
│
├── scripts/
│   ├── pre_test.sh
│   └── post_test.sh
│
├── .gitignore
├── README.md
├── requirements.txt
└── robot_settings.yaml
```

Explanation of Each Component

1. tests/:
- test_suites/: Contains directories for different test suites, each with its own set of test cases.
  - suite1/, suite2/: Example test suites.
    - **test_case1.robot**, **test_case2.robot**: Individual test case files for each suite.
- **test_cases/**: Contains standalone test cases that are not part of any specific suite.
  - **test_case1.robot**, **test_case2.robot**: Individual test case files.

2. resources/:
- **keywords/**: Contains reusable keyword definitions.
  - custom_keywords.robot: Example file for custom keywords.
- variables/: Contains variable definitions used across test cases.
  - common_variables.robot: Example file for common variables.
- locators/: Contains locators for web elements.
  - page_locators.robot: Example file for page element locators.

3. libraries/:
- custom_library.py: Custom Python libraries that extend Robot Framework functionalities.

4. results/:
- log.html, report.html, output.xml: Test execution results and logs generated by Robot Framework.

5. configs/:
- config1.py, config2.py: Configuration files for different environments or settings.

6. logs/:
- debug.log, errors.log: Logs for debugging and error tracking.

7. data/:
- test_data.xlsx: Data files used for data-driven testing.

8. scripts/:
- pre_test.sh, post_test.sh: Shell scripts for setup and teardown operations before and after tests.

9. .gitignore:
- Specifies files and directories to be ignored by Git.

10. README.md:
- Documentation and overview of the project.

11. requirements.txt:
- Lists Python dependencies required for the project.

12. robot_settings.yaml:
- Configuration file for Robot Framework settings (optional, depending on the project setup).

Best Practices for Folder Structure

- **Modularity:** Keep test cases, keywords, variables, and locators in separate directories to promote reusability and ease of maintenance.
- **Clarity:** Use descriptive names for directories and files to make the structure self-explanatory.
- **Scalability:** Design the structure to accommodate growth, allowing easy addition of new test suites, test cases, and resources.
- **Version Control:** Use a `.gitignore` file to exclude unnecessary files and directories from version control.

This folder structure ensures that your Robot Framework project remains organized, making it easier to manage, extend, and collaborate with other team members.

Setting Up Your Environment

First, ensure you have Robot Framework installed and set up in your environment. You can install it using pip:

```bash
pip install robotframework
```

Creating Custom Keywords in Python

1. Import Required Modules:

   ```python
   from robot.api.deco import keyword
   ```

2. Custom Keyword Examples:

   Here are 50 examples of custom keywords using the `@keyword` decorator:

   ```python
   from robot.api.deco import keyword

   class CustomKeywords:
   ```

```
@keyword
def log_hello_world(self):
    print("Hello, World!")

@keyword
def sum_two_numbers(self, num1, num2):
    return int(num1) + int(num2)

@keyword
def multiply_two_numbers(self, num1, num2):
    return int(num1) * int(num2)

@keyword
def subtract_two_numbers(self, num1, num2):
    return int(num1) - int(num2)

@keyword
def divide_two_numbers(self, num1, num2):
    return int(num1) / int(num2)

@keyword
def concatenate_strings(self, str1, str2):
    return str1 + str2

@keyword
def get_string_length(self, string):
    return len(string)

@keyword
def convert_to_uppercase(self, string):
    return string.upper()

@keyword
def convert_to_lowercase(self, string):
    return string.lower()

@keyword
def is_string_palindrome(self, string):
    return string == string[::-1]

@keyword
def sort_list(self, *args):
    return sorted(args)

@keyword
def find_max_in_list(self, *args):
    return max(args)
```

```
@keyword
def find_min_in_list(self, *args):
    return min(args)

@keyword
def reverse_list(self, *args):
    return list(reversed(args))

@keyword
def list_contains(self, item, *args):
    return item in args

@keyword
def find_element_index(self, item, *args):
    return args.index(item)

@keyword
def add_element_to_list(self, item, *args):
    return list(args) + [item]

@keyword
def remove_element_from_list(self, item, *args):
    args_list = list(args)
    args_list.remove(item)
    return args_list

@keyword
def count_occurrences(self, item, *args):
    return args.count(item)

@keyword
def join_list_elements(self, separator, *args):
    return separator.join(args)

@keyword
def split_string(self, separator, string):
    return string.split(separator)

@keyword
def replace_substring(self, old, new, string):
    return string.replace(old, new)

@keyword
def starts_with(self, prefix, string):
    return string.startswith(prefix)

@keyword
def ends_with(self, suffix, string):
    return string.endswith(suffix)

@keyword
def is_substring(self, substring, string):
    return substring in string

@keyword
def get_substring(self, start, end, string):
    return string[start:end]

@keyword
def round_number(self, number, digits):
    return round(float(number), int(digits))

@keyword
def is_even_number(self, number):
    return int(number) % 2 == 0

@keyword
def is_odd_number(self, number):
    return int(number) % 2 != 0

@keyword
def calculate_factorial(self, number):
    if int(number) == 0:
        return 1
    else:
        return int(number) * self.calculate_factorial(int(number) - 1)

@keyword
def find_fibonacci_sequence(self, n):
    n = int(n)
    fib_sequence = [0, 1]
    for i in range(2, n):
        fib_sequence.append(fib_sequence[-1] + fib_sequence[-2])
    return fib_sequence

@keyword
def check_prime(self, number):
    number = int(number)
    if number > 1:
        for i in range(2, int(number/2)+1):
            if (number % i) == 0:
                return False
        return True
    else:
        return False
```

```
@keyword
def find_prime_numbers(self, start, end):
    primes = []
    for num in range(int(start), int(end) + 1):
        if self.check_prime(num):
            primes.append(num)
    return primes

@keyword
def sort_dictionary_by_keys(self, dictionary):
    return dict(sorted(dictionary.items()))

@keyword
def sort_dictionary_by_values(self, dictionary):
    return dict(sorted(dictionary.items(), key=lambda item: item[1]))

@keyword
def merge_dictionaries(self, dict1, dict2):
    result = dict1.copy()
    result.update(dict2)
    return result

@keyword
def find_keys_in_dict(self, dictionary):
    return list(dictionary.keys())

@keyword
def find_values_in_dict(self, dictionary):
    return list(dictionary.values())

@keyword
def get_value_from_dict(self, key, dictionary):
    return dictionary.get(key, None)

@keyword
def add_key_value_to_dict(self, key, value, dictionary):
    dictionary[key] = value
    return dictionary

@keyword
def remove_key_from_dict(self, key, dictionary):
    dictionary.pop(key, None)
    return dictionary

@keyword
def check_key_exists(self, key, dictionary):
    return key in dictionary
@keyword
def find_intersection_of_sets(self, set1, set2):
    return set1.intersection(set2)

@keyword
def find_union_of_sets(self, set1, set2):
    return set1.union(set2)

@keyword
def find_difference_of_sets(self, set1, set2):
    return set1.difference(set2)

@keyword
def is_subset(self, subset, superset):
    return subset.issubset(superset)

@keyword
def is_superset(self, superset, subset):
    return superset.issuperset(subset)
```

Usage in Robot Framework Test Cases

1. Create a Python file with the custom keywords:

   ```bash
   touch custom_keywords.py
   ```

   Copy the `CustomKeywords` class into `custom_keywords.py`.

2. Import and Use Custom Keywords in Your Test Suite:

```robot
*** Settings ***
Library    custom_keywords.py

*** Test Cases ***
Test Log Hello World
    Log Hello World

Test Sum Two Numbers
    ${result}    Sum Two Numbers    5    7
    Should Be Equal    ${result}    12

Test Multiply Two Numbers
    ${result}    Multiply Two Numbers    3    4
```

```
        Should Be Equal    ${result}    12

    # Add other test cases for remaining keywords similarly
    ```

Running the Tests

Execute the Robot Framework tests using the following command:

```bash
robot path_to_your_test_suite.robot
```

These examples should cover a wide range of functionalities and provide a solid foundation for creating custom keywords using the **`robot.api.deco`** module in Python for Robot Framework.

Prerequisites
Ensure you have `robotframework` installed:
```sh
pip install robotframework
```

Directory Structure
Create the following structure:
```
.
├── keywords
│   ├── __init__.py
│   ├── file_operations.py
│   ├── string_operations.py
│   └── web_operations.py
├── tests
│   └── test_sample_keywords.robot
└── requirements.txt
```

Sample Keywords

# `keywords/file_operations.py`

```python
from robot.api.deco import keyword

class FileOperations:

    @keyword
    def create_file(self, file_path, content):
```

```
        with open(file_path, 'w') as f:
            f.write(content)

    @keyword
    def read_file(self, file_path):
        with open(file_path, 'r') as f:
            return f.read()

    @keyword
    def delete_file(self, file_path):
        import os
        os.remove(file_path)

    @keyword
    def append_to_file(self, file_path, content):
        with open(file_path, 'a') as f:
            f.write(content)

    @keyword
    def file_should_exist(self, file_path):
        import os
        if not os.path.exists(file_path):
            raise AssertionError(f"File '{file_path}' does not exist.")

    @keyword
    def file_should_not_exist(self, file_path):
        import os
        if os.path.exists(file_path):
            raise AssertionError(f"File '{file_path}' should not exist.")
```

# `keywords/string_operations.py`

```python
from robot.api.deco import keyword

class StringOperations:

    @keyword
    def concatenate_strings(self, *args):
        return ''.join(args)

    @keyword
    def split_string(self, string, delimiter):
        return string.split(delimiter)

    @keyword
    def string_should_contain(self, string, substring):
        if substring not in string:
            raise AssertionError(f"'{string}' does not contain '{substring}'.")

    @keyword
    def string_should_not_contain(self, string, substring):
        if substring in string:
            raise AssertionError(f"'{string}' should not contain '{substring}'.")

    @keyword
    def convert_to_uppercase(self, string):
        return string.upper()

    @keyword
    def convert_to_lowercase(self, string):
        return string.lower()
```

# `keywords/web_operations.py`

```python
from robot.api.deco import keyword
from selenium import webdriver

class WebOperations:

    def __init__(self):
        self.driver = None

    @keyword
    def open_browser(self, url, browser='chrome'):
        if browser == 'chrome':
            self.driver = webdriver.Chrome()
        elif browser == 'firefox':
            self.driver = webdriver.Firefox()
        else:
            raise ValueError(f"Unsupported browser: {browser}")
        self.driver.get(url)

    @keyword
    def close_browser(self):
        if self.driver:
            self.driver.quit()

    @keyword
    def click_element(self, locator):
        self.driver.find_element_by_css_selector(locator).click()

    @keyword
    def input_text(self, locator, text):
        element = self.driver.find_element_by_css_selector(locator)
        element.clear()
        element.send_keys(text)

    @keyword
    def element_should_contain_text(self, locator, text):
        element = self.driver.find_element_by_css_selector(locator)
        if text not in element.text:
            raise AssertionError(f"Element '{locator}' does not contain text '{text}'.")

    @keyword
    def element_should_be_visible(self, locator):
        element = self.driver.find_element_by_css_selector(locator)
        if not element.is_displayed():
            raise AssertionError(f"Element '{locator}' is not visible.")
```

Combining All Keywords

# `keywords/__init__.py`

```python
from .file_operations import FileOperations
from .string_operations import StringOperations
from .web_operations import WebOperations

file_operations = FileOperations()
string_operations = StringOperations()
web_operations = WebOperations()
```

Using Keywords in Robot Framework

# `tests/test_sample_keywords.robot`

```robot
*** Settings ***
Library    keywords/file_operations.py
Library    keywords/string_operations.py
Library    keywords/web_operations.py

*** Test Cases ***
Test File Operations
    Create File    test.txt    This is a test file.
    File Should Exist    test.txt
    ${content}=    Read File    test.txt
    Should Be Equal    ${content}    This is a test file.
    Append To File    test.txt    Appended text.
    ${content}=    Read File    test.txt
    Should Be Equal    ${content}    This is a test file.Appended text.
    Delete File    test.txt
    File Should Not Exist    test.txt

Test String Operations
    ${result}=    Concatenate Strings    Hello    World
    Should Be Equal    ${result}    HelloWorld
    ${parts}=    Split String    Hello,World    ,
    Should Be Equal As Strings    ${parts[0]}    Hello
    Should Be Equal As Strings    ${parts[1]}    World
    String Should Contain    HelloWorld    World
    String Should Not Contain    HelloWorld    Foo
    ${upper}=    Convert To Uppercase    hello
    Should Be Equal    ${upper}    HELLO
    ${lower}=    Convert To Lowercase    HELLO
    Should Be Equal    ${lower}    hello
```

Test Web Operations

```
    Open Browser    http://example.com
    Element Should Be Visible    h1
    Close Browser
```

Explanation

- File Operations: Custom keywords for creating, reading, appending, and deleting files.
- String Operations: Custom keywords for manipulating and validating strings.
- Web Operations: Custom keywords for interacting with web elements using Selenium WebDriver.

This setup provides a comprehensive example of how to define and use custom keywords in Robot Framework using Python and the `robot.api.deco` decorators. You can expand this further based on specific testing requirements.