Cypress

Basic Cypress Concepts

1. What is Cypress, and how does it differ from Selenium?

- **Cypress**: A modern, JavaScript-based end-to-end testing framework primarily designed for web applications. It operates directly in the browser, providing faster execution and better debugging capabilities.
- **Key Differences**:
    - **Execution**: Cypress runs directly in the browser, while Selenium uses a WebDriver to interact with browsers externally.
    - **Asynchronous Handling**: Cypress natively handles asynchronous operations; Selenium requires explicit waits.
    - **Setup**: Cypress is easier to install and configure compared to Selenium.
    - **Use Cases**: Selenium supports multiple languages and browsers, while Cypress focuses on JavaScript and Chromium-based browsers.

---

2. What are the key features of Cypress?

- Real-time reloading.
- Automatic waits for elements.
- Built-in debugging.
- Time travel to see commands in action.
- Network stubbing and mocking.
- In-browser testing.

---

3. What are the prerequisites for installing and running Cypress?

- Node.js installed on the system.
- A project directory initialized with `npm init`.
- A supported browser (e.g., Chrome, Firefox).
- Knowledge of JavaScript.

---

4. What type of applications can you test using Cypress?

- Web applications (Single Page Applications or traditional websites).
- Progressive Web Apps (PWAs).
- Applications built with frameworks like React, Angular, and Vue.js.

---

5. Explain the Cypress folder structure after installation.

- **cypress/integration**: Stores test files.
- **cypress/fixtures**: Holds static data files (e.g., JSON).
- **cypress/plugins**: Used for extending Cypress functionalities.
- **cypress/support**: Contains reusable utilities like commands.
- **cypress/videos/screenshots**: Stores test execution media.

---

6. How do you configure Cypress for a project?

Add Cypress to the project using:

bash
Copy code
npm install cypress --save-dev

-

Open Cypress:

bash
Copy code
npx cypress open

-

Configure in `cypress.config.js` (e.g., base URL, timeouts):

javascript
Copy code
```
module.exports = {
  e2e: {
    baseUrl: '<http://example.com>',
    defaultCommandTimeout: 10000,
  },
};
```

-

Core Cypress Functionality

1. How do you create a test case in Cypress?

Example:

```
 javascript
Copy code
describe('Login Test', () => {
  it('Should log in successfully', () => {
    cy.visit('<http://example.com/login>');
    cy.get('#username').type('user');
    cy.get('#password').type('password');
    cy.get('#loginButton').click();
    cy.url().should('include', '/dashboard');
  });
});
```

-

---

2. Explain the Cypress command execution order.

- Cypress commands are asynchronous and executed in the order they are written. Cypress queues commands and processes them sequentially.

Example:

```
 javascript
Copy code
cy.visit('/home');  // Executed first
cy.get('#button').click();  // Executed next
```

-

---

3. What are Cypress commands, and how do you chain them?

- Cypress commands are built-in methods (e.g., `cy.visit`, `cy.get`) used for interacting with and asserting on web elements.

Chaining Example:

javascript
Copy code
```
cy.get('#searchBox').type('Cypress Testing').should('have.value', 'Cypress Testing');
```

-

---

4. How do you handle assertions in Cypress?

- Cypress provides BDD-style assertions:

`should`:

javascript
Copy code
```
cy.get('#element').should('be.visible');
```

    -

`expect`:

javascript
Copy code
```
expect(true).to.be.true;
```

    -

`assert`:

javascript
Copy code
```
assert.equal(2, 2, 'Values are equal');
```

-

---

5. How do you handle waits and timeouts in Cypress?

Cypress automatically waits for DOM elements. Custom waits can be added using:

javascript
Copy code
```
cy.wait(5000); // Wait for 5 seconds
```

-

---

6. How does Cypress manage asynchronous operations?

- Cypress automatically retries commands until they pass or time out. It uses Promises to handle asynchronous operations internally.

---

7. How do you interact with elements in Cypress?

- Example:

Click:

javascript
Copy code
```
cy.get('#button').click();
```

-

Type:

javascript
Copy code
```
cy.get('#input').type('Sample Text');
```

-

Select:

```
javascript
Copy code
cy.get('#dropdown').select('Option 1');
```

- ○

---

8. How do you handle iframes in Cypress?

Use `cy.iframe()` from the `cypress-iframe` plugin:

```
javascript
Copy code
cy.frameLoaded('#iframe');
cy.iframe().find('#button').click();
```

- •

Cypress Testing Features

1. What are fixtures in Cypress, and how do you use them?

- Fixtures are static data files (e.g., JSON) used for testing.

Example:

```
javascript
Copy code
cy.fixture('data.json').then((data) => {
  cy.get('#username').type(data.username);
});
```

- •

---

2. How do you perform API testing using Cypress?

Example:

```
 javascript
Copy code
cy.request('GET', '/api/users').then((response) => {
  expect(response.status).to.eq(200);
});
```

-

---

3. What are Cypress hooks, and how do you use them?

- Hooks are lifecycle methods:
    - `before`: Runs once before all tests.
    - `beforeEach`: Runs before every test.

Example:

```
 javascript
Copy code
beforeEach(() => {
  cy.visit('/login');
});
```

-

Advanced Concepts

1. How do you implement data-driven testing in Cypress?

Example:

```
 javascript
Copy code
const users = [
  { username: 'user1', password: 'pass1' },
  { username: 'user2', password: 'pass2' },
];
```

```
users.forEach((user) => {
  it(`Logs in with ${user.username}`, () => {
    cy.get('#username').type(user.username);
    cy.get('#password').type(user.password);
    cy.get('#loginButton').click();
  });
});
```

-

---

2. How do you mock or stub network requests in Cypress?

Example:

```
 javascript
Copy code
cy.intercept('GET', '/api/users', { fixture: 'users.json' }).as('getUsers');
cy.visit('/users');
cy.wait('@getUsers');
```

-

Best Practices

1. **Use reusable commands** in `cypress/support/commands.js`.
2. **Organize tests** in logical folders.
3. **Avoid hardcoding** sensitive data; use environment variables.

Scenario-Based Questions

1. How would you write a test for a login functionality?

Example:

```
 javascript
Copy code
cy.visit('/login');
```

```
cy.get('#username').type('user');
cy.get('#password').type('password');
cy.get('#loginButton').click();
cy.url().should('include', '/dashboard');
```

-

2. How do you validate a search functionality?

Example:

```
 javascript
Copy code
cy.get('#searchBox').type('Cypress').type('{enter}');
cy.get('.results').should('contain', 'Cypress Testing');
```

-

Detailed explaination

Basic Cypress Script Explanation

Scenario: Writing a login test.

Test Script:

```
 javascript
Copy code
describe('Login Test Suite', () => {

  // Runs before all tests in the block
  before(() => {
    cy.log('Starting Login Test Suite');
  });

  // Runs before each individual test
  beforeEach(() => {
    cy.visit('<https://example.com/login>'); // Step 1: Visit the login page
  });

  it('Should log in successfully with valid credentials', () => {
    cy.get('#username') // Step 2: Find the username input field by its ID
```

```
    .type('validUser'); // Step 3: Type in a valid username

  cy.get('#password') // Step 4: Find the password input field by its ID
    .type('validPassword'); // Step 5: Type in a valid password

  cy.get('#loginButton') // Step 6: Find the login button
    .click(); // Step 7: Click the login button

  cy.url() // Step 8: Get the current URL
    .should('include', '/dashboard'); // Step 9: Assert that the URL includes '/dashboard'
 });

 it('Should show error for invalid credentials', () => {
  cy.get('#username').type('invalidUser'); // Type an invalid username
  cy.get('#password').type('invalidPassword'); // Type an invalid password
  cy.get('#loginButton').click(); // Click the login button

  cy.get('.error-message') // Step 10: Find the error message element
    .should('be.visible') // Step 11: Assert that the error message is visible
    .and('contain', 'Invalid username or password'); // Step 12: Assert the message text
 });

});
```

- ---

  ## Explanation of the Script

  - **Test Suite (`describe`)**:
    - Groups related test cases for better organization.
    - In this case, all tests related to the "Login" functionality are grouped.

```
javascript
Copy code
describe('Login Test Suite', () => {
```

  -
    - **Why use `describe`?** It provides a logical structure to tests, making it easier to read and debug.
- ---
  - **`before` Hook**:

        - Runs once before all test cases in the suite.
        - Can be used for setting up global configurations or logging information.

javascript
Copy code
```
before(() => {
  cy.log('Starting Login Test Suite');
});
```

    -

- ---
    - **`beforeEach` Hook**:

        - Runs before each test case.
        - Here, it navigates to the login page before executing each test.

javascript
Copy code
```
beforeEach(() => {
  cy.visit('<https://example.com/login>');
});
```

    -

        - **Why use `beforeEach`?** Ensures a consistent starting point for every test.

- ---
    - **Test Case (`it`)**:

        - Represents a single, isolated functionality being tested.
        - For example, the first `it` block tests successful login, while the second checks for error handling.

javascript
Copy code
```
it('Should log in successfully with valid credentials', () => {
```

    -

- ---
    - **Commands**:

        - Cypress commands like `cy.get`, `cy.type`, and `cy.click` interact with web elements.

javascript
Copy code
```
cy.get('#username') // Finds the username input by ID
  .type('validUser'); // Types "validUser" into the field
```

    -
        - **Why use `cy.get`?** It's used to select elements on the page, similar to how `document.querySelector` works in JavaScript.
- ---
    - **Assertions**:
        - `should` and `and` are used to validate test outcomes.

javascript
Copy code
```
cy.url().should('include', '/dashboard');
```

    -
        - **What does this do?** It checks that after login, the URL contains `/dashboard`, which is a common way to confirm successful navigation.

javascript
Copy code
```
cy.get('.error-message').should('be.visible');
```

    -
        - **Why assert visibility?** Ensures the user sees an error when providing incorrect login credentials.

Scenario-Based Example: Testing Search Functionality

Scenario: Validate a search feature on an e-commerce website.

Test Script:

javascript
Copy code

```
describe('Search Functionality Test Suite', () => {

  beforeEach(() => {
    cy.visit('<https://example-ecommerce.com>'); // Step 1: Navigate to the homepage
  });

  it('Should return search results for valid input', () => {
    cy.get('#searchBox') // Step 2: Find the search input box
      .type('Laptop'); // Step 3: Type "Laptop" into the search box

    cy.get('#searchButton') // Step 4: Find the search button
      .click(); // Step 5: Click the search button

    cy.get('.results') // Step 6: Locate the results container
      .should('be.visible') // Step 7: Assert results are visible
      .and('contain', 'Laptop'); // Step 8: Assert that results contain the keyword "Laptop"
  });

  it('Should display no results message for invalid input', () => {
    cy.get('#searchBox').type('RandomNonExistentProduct'); // Type an invalid product name
    cy.get('#searchButton').click();
    cy.get('.no-results') // Locate the no-results message element
      .should('be.visible') // Assert that it is visible
      .and('contain', 'No products found'); // Assert the correct message is displayed
  });

});
```

- ---

  ## Explanation of the Script

  - **Setup**:
    - Visits the homepage before each test to ensure a fresh start.

```
javascript
Copy code
beforeEach(() => {
  cy.visit('<https://example-ecommerce.com>');
});
```

  -

- ---
    - **Valid Search**:
        - Inputs "Laptop" in the search box and checks that the results are visible and relevant.

javascript
Copy code
```
cy.get('#searchBox').type('Laptop');
cy.get('#searchButton').click();
cy.get('.results').should('be.visible').and('contain', 'Laptop');
```

    -
- ---
    - **Invalid Search**:
        - Inputs an invalid product name and checks for the "No products found" message.

javascript
Copy code
```
cy.get('#searchBox').type('RandomNonExistentProduct');
cy.get('#searchButton').click();
cy.get('.no-results').should('be.visible').and('contain', 'No products found');
```

    -
- ---
    ## Debugging Tests
    - **Use `cy.log`**:
        - Insert custom logs to track test flow.

javascript
Copy code
```
cy.log('Typing username');
```

    -
    - **Command Log**:
        - Use the Cypress GUI to inspect each command and its result.

- **Screenshots and Videos**:
  - Automatically captured on failure, stored in `cypress/screenshots`.

- ---

Best Practices

- **Use Reusable Commands**:

Define common actions in `cypress/support/commands.js`:

```
 javascript
Copy code
Cypress.Commands.add('login', (username, password) => {
  cy.get('#username').type(username);
  cy.get('#password').type(password);
  cy.get('#loginButton').click();
});
```

- **Avoid Hardcoding**:

Use environment variables for sensitive data:

```
 javascript
Copy code
cy.login(Cypress.env('username'), Cypress.env('password'));
```

- **Follow Page Object Model (POM)**:
  - Organize selectors and actions in separate files for maintainability.

- Step by Step POM in cypress

Page Object Model (POM) in Cypress: Step-by-Step Guide with Examples

What is the Page Object Model (POM)?

- **Definition**: POM is a design pattern used in test automation to create an object repository for web elements. It separates the UI (Page Objects) from the test scripts to enhance **reusability**, **maintainability**, and **readability**.
- **Analogy**: Think of it as a blueprint where each page of your application has its own class or object.

Step-by-Step Notes for POM in Cypress

1. Why Use POM?

- Enhances **test reusability** by centralizing UI elements.
- Reduces code duplication across tests.
- Makes tests easier to **maintain** if UI changes occur.
- Encourages better **code organization**.

---

2. Setting Up POM in Cypress

Step 1: Organize the Folder Structure

- Create a folder called `pages` inside the `cypress` directory.
- Each page (e.g., Login Page, Dashboard Page) will have its own file.

Example Folder Structure:

```
 Copy code
cypress/
  ├── integration/
  │   └── login.spec.js
  ├── pages/
  │   ├── LoginPage.js
  │   └── DashboardPage.js
  ├── support/
  │   └── commands.js
```

Step 2: Define a Page Object

- Each page object contains:
  - Selectors for the page elements.
  - Methods for interacting with the elements.

Example: LoginPage.js

```
 javascript
Copy code
class LoginPage {
  // Define selectors
  usernameField = '#username';
  passwordField = '#password';
  loginButton = '#loginButton';

  // Define methods
  enterUsername(username) {
    cy.get(this.usernameField).type(username); // Action: Type in username
  }

  enterPassword(password) {
    cy.get(this.passwordField).type(password); // Action: Type in password
  }

  clickLoginButton() {
    cy.get(this.loginButton).click(); // Action: Click login button
  }
}

// Export the class to use in test scripts
export default new LoginPage();
```

Step 3: Use the Page Object in a Test

Test Script: login.spec.js

```
 javascript
Copy code
import LoginPage from '../pages/LoginPage'; // Import the LoginPage

describe('Login Page Test Suite', () => {
  it('Should log in successfully with valid credentials', () => {
    cy.visit('<https://example.com/login>'); // Navigate to the login page

    // Use methods from LoginPage
    LoginPage.enterUsername('validUser');
    LoginPage.enterPassword('validPassword');
    LoginPage.clickLoginButton();
```

```
    // Assert successful login
    cy.url().should('include', '/dashboard');
  });
});
```

- ---

3. Adding Reusability

    - Page Objects should encapsulate **only UI interactions**. For common flows, create reusable methods.

Example: Adding a Login Flow in LoginPage.js

```
 javascript
Copy code
class LoginPage {
  usernameField = '#username';
  passwordField = '#password';
  loginButton = '#loginButton';

  login(username, password) {
    cy.get(this.usernameField).type(username);
    cy.get(this.passwordField).type(password);
    cy.get(this.loginButton).click();
  }
}

export default new LoginPage();
```

Test Script:

```
 javascript
Copy code
LoginPage.login('validUser', 'validPassword'); // Reuse login flow
```

- ---

4. Handling Multiple Pages

    - Create a separate page object for each page.

Example: DashboardPage.js

```
 javascript
Copy code
class DashboardPage {
  welcomeMessage = '.welcome-message';

  verifyWelcomeMessage() {
    cy.get(this.welcomeMessage).should('contain', 'Welcome, User');
  }
}

export default new DashboardPage();
```

Test Script:

```
 javascript
Copy code
import DashboardPage from '../pages/DashboardPage';

DashboardPage.verifyWelcomeMessage(); // Assert welcome message
```

- ---

  ## 5. Best Practices for POM

  - **Single Responsibility**:
    - Each page object should handle only its specific elements and actions.
  - **Encapsulation**:
    - Keep locators private and expose only meaningful methods.
  - **Reusability**:
    - Create reusable methods for common flows like login, logout, etc.
  - **Descriptive Names**:
    - Use meaningful names for methods and selectors.
  - **Avoid Hardcoding**:
    - Use environment variables or config files for data like URLs.

Example Project

Scenario: Test Search Functionality

SearchPage.js

```
 javascript
Copy code
class SearchPage {
  searchBox = '#searchBox';
  searchButton = '#searchButton';
  results = '.results';

  performSearch(keyword) {
    cy.get(this.searchBox).type(keyword);
    cy.get(this.searchButton).click();
  }

  verifySearchResults(expectedText) {
    cy.get(this.results).should('contain', expectedText);
  }
}

export default new SearchPage();
```

Test Script:

```
 javascript
Copy code
import SearchPage from '../pages/SearchPage';

describe('Search Functionality Test', () => {
  it('Should display results for valid search', () => {
    cy.visit('<https://example-ecommerce.com>');
    SearchPage.performSearch('Laptop');
    SearchPage.verifySearchResults('Laptop');
  });
});
```

- ---

  ## Interview Questions and Answers

  - **What is POM in Cypress, and why is it important?**
    - **Answer**: POM is a design pattern that organizes UI elements and actions into separate classes, making test scripts more maintainable and reusable. It is important because it helps manage changes to the UI efficiently and improves test readability.

- ---

- **How do you structure a POM framework in Cypress?**
  - **Answer**: Create a `pages` folder in the Cypress directory. Each page of the application gets its own class file containing locators and methods. Test scripts import these classes and use the methods for testing.

- **What are the benefits of using POM?**
  - Reduces code duplication.
  - Improves test maintainability.
  - Encapsulates UI logic, separating it from test logic.

- **Can POM be integrated with data-driven testing in Cypress?**
  - **Answer**: Yes, Cypress can use fixtures or external data files to drive tests. Combine POM methods with data from fixtures to create reusable, data-driven tests.

Example:

```
 javascript
Copy code
cy.fixture('userData').then((data) => {
  LoginPage.login(data.username, data.password);
});
```

- Step by Step BDD Cucumber in Cypress

BDD Cucumber Framework with Cypress

What is BDD?

- **BDD (Behavior-Driven Development)** is a development approach that encourages collaboration between developers, testers, and business stakeholders by using simple, human-readable language for test scenarios.
- **Cucumber** is a tool that enables BDD by allowing test cases to be written in **Gherkin syntax** (plain English text).

Core Concepts of BDD and Cucumber

- **Feature Files**: Define test scenarios in Gherkin syntax.
- **Step Definitions**: Link Gherkin steps to automation code.
- **Cypress and Cucumber Integration**: Automate BDD scenarios using Cypress commands.

Step-by-Step Notes for Implementing BDD with Cucumber

1. Install Required Packages

Run the following commands to install Cypress and Cucumber dependencies:

bash
Copy code
npm install cypress @badeball/cypress-cucumber-preprocessor @cucumber/cucumber --save-dev

- ### 2. Configure Cypress for Cucumber

Update `cypress.config.js`: Add the Cucumber preprocessor plugin.

javascript
Copy code
```
const { defineConfig } = require("cypress");
const createBundler = require("@bahmutov/cypress-esbuild-preprocessor");
const addCucumberPreprocessorPlugin = require("@badeball/cypress-cucumber-preprocessor").addCucumberPreprocessorPlugin;
const createEsbuildPlugin = require("@badeball/cypress-cucumber-preprocessor/esbuild");

module.exports = defineConfig({
  e2e: {
    setupNodeEvents(on, config) {
      const bundler = createBundler({
        plugins: [createEsbuildPlugin(config)],
      });

      on("file:preprocessor", bundler);
      addCucumberPreprocessorPlugin(on, config);

      return config;
    },
    specPattern: "cypress/e2e/**/*.feature",
  },
});
```

-

Set Spec Pattern: Ensure Cypress recognizes `.feature` files:

javascript
Copy code
specPattern: "cypress/e2e/**/*.feature",

-

---

3. Create Folder Structure

Organize your project for BDD:

Copy code
```
cypress/
  ├── e2e/
  │   ├── features/
  │   │   └── login.feature
  │   ├── step_definitions/
  │   │   └── loginSteps.js
  ├── support/
  │   └── commands.js
```

---

4. Write a Feature File

Feature: Describes the test scenario using Gherkin syntax.

Example: `login.feature`

gherkin
Copy code
```
Feature: Login Functionality

  Scenario: Successful login with valid credentials
    Given I open the login page
    When I enter valid credentials
    Then I should be redirected to the dashboard
```

---

5. Create Step Definitions

Step Definitions: Map Gherkin steps to Cypress commands.

Example: `loginSteps.js`

javascript
Copy code

```
import { Given, When, Then } from "@badeball/cypress-cucumber-preprocessor";

Given("I open the login page", () => {
  cy.visit("<https://example.com/login>");
});

When("I enter valid credentials", () => {
  cy.get("#username").type("validUser");
  cy.get("#password").type("validPassword");
  cy.get("#loginButton").click();
});

Then("I should be redirected to the dashboard", () => {
  cy.url().should("include", "/dashboard");
});
```

---

6. Run the Tests

Execute tests using the Cypress Test Runner:

bash
Copy code

```
npx cypress open
```

- Choose the `login.feature` file from the Cypress GUI to run the BDD test.

Notes for Students

- **Gherkin Syntax Overview**:
  - **Feature**: A high-level description of functionality.
  - **Scenario**: A specific example of the feature.

        - **Steps**: Keywords like `Given`, `When`, `Then` describe the scenario.
    - **Advantages of BDD**:
        - Improves collaboration and communication.
        - Scenarios are written in plain English, making them accessible to non-technical stakeholders.
        - Promotes better understanding of business requirements.

Writing Reusable Steps: Use parameterization to reuse steps across scenarios.

Example:

gherkin
Copy code
Scenario: Login with multiple credentials
When I log in with username "user1" and password "pass1"
Then I should see the dashboard

When I log in with username "user2" and password "pass2"
Then I should see the dashboard

Step Definition:

javascript
Copy code
When("I log in with username {string} and password {string}", (username, password) => {
cy.get("#username").type(username);
cy.get("#password").type(password);
cy.get("#loginButton").click();
});

    -

- ---

Interview Questions and Answers

    - **What is BDD, and how does Cucumber support it?**
        - **Answer**: BDD is a development approach that uses natural language to describe test scenarios. Cucumber supports BDD by allowing tests to be written in Gherkin syntax and executed using automation tools like Cypress.
    - **How do you integrate Cucumber with Cypress?**

        - **Answer**: Install the necessary plugins (`@badeball/cypress-cucumber-preprocessor`), configure the `cypress.config.js`, and write `.feature` files for scenarios.
    - **What is the purpose of Step Definitions in BDD?**
        - **Answer**: Step Definitions map Gherkin steps to automation logic, linking the plain English scenarios to Cypress commands.
    - **How do you handle dynamic data in BDD scenarios?**
        - **Answer**: Use parameters in Gherkin steps and pass them to Step Definitions.

Example:

gherkin
Copy code
Given I search for "{string}"

Step Definition:

javascript
Copy code
```
Given("I search for {string}", (searchTerm) => {
  cy.get("#searchBox").type(searchTerm);
});
```

    -
    - **What are the limitations of Cucumber in Cypress?**
        - **Answer**:
            - It adds complexity if not needed for the project.
            - Scenarios can become verbose and hard to manage if poorly structured.
            - Limited support for non-browser testing compared to other tools.

Sample Project

Feature: Search Functionality

File: `search.feature`

gherkin
Copy code
```
Feature: Search functionality on e-commerce site

  Scenario: Search for a product
    Given I am on the home page
    When I search for "Laptop"
    Then I should see results related to "Laptop"
```

File: `searchSteps.js`

javascript
Copy code
```
import { Given, When, Then } from "@badeball/cypress-cucumber-preprocessor";

Given("I am on the home page", () => {
  cy.visit("<https://example-ecommerce.com>");
});

When("I search for {string}", (product) => {
  cy.get("#searchBox").type(product);
  cy.get("#searchButton").click();
});

Then("I should see results related to {string}", (product) => {
  cy.get(".results").should("contain", product);
});
```

- **Execution**: Run the `search.feature` file using Cypress.

Final Notes

- BDD is great for projects involving collaboration between teams.
- Use **Cucumber with Cypress** for clear, maintainable, and reusable test cases.
- Start small with simple scenarios and gradually introduce more complex flows to understand the framework thoroughly.

Cypress Basics

1. **What is Cypress, and why is it used for testing?**

- **Definition**: Cypress is a JavaScript-based end-to-end testing framework designed for modern web applications. It enables developers and QA engineers to write fast, reliable, and maintainable tests for web applications.
- **Why Use Cypress**:
  - Supports automatic waiting.
  - Provides time travel to debug test failures.
  - Runs directly in the browser for more accurate testing.
  - Includes built-in support for stubbing and spying on network requests.

---

1. **What types of applications can be tested using Cypress?**
   - Cypress can test any application that runs in a web browser. This includes:
     - Single Page Applications (SPA).
     - Progressive Web Applications (PWA).
     - Applications built with frameworks like React, Angular, Vue, etc.

---

1. **What are the key advantages of Cypress over other frameworks?**
   - Real-time reloading and debugging.
   - Built-in wait for elements and DOM updates.
   - Direct control of browser behavior via the browser process.
   - Easy to set up without additional dependencies.

---

1. **What are the limitations of Cypress?**
   - No support for testing multiple tabs or multiple browsers in the same test.
   - Limited cross-browser support compared to Selenium.
   - Cannot run tests on older browsers like Internet Explorer.
   - Testing native mobile applications is not supported.

Core Cypress Concepts

1. **How does Cypress work under the hood?**
   - Cypress operates in the same run loop as the application, meaning it runs directly inside the browser.
   - This architecture allows Cypress to intercept requests, manipulate DOM, and understand browser events in real time.

---

1. **What are Cypress commands? Provide examples of commonly used commands.**

- **Cypress Commands**: Actions used to interact with or test the state of a web application.
- **Examples**:
  - `cy.visit('/login')`: Navigate to the login page.
  - `cy.get('#username').type('user123')`: Enter text into an input field.
  - `cy.contains('Submit').click()`: Click a button with specific text.

---

1. **What is the difference between `cy.get()` and `cy.contains()`?**
   - `cy.get()`: Selects elements based on a CSS selector.
     - Example: `cy.get('.btn')` selects elements with the class `btn`.
   - `cy.contains()`: Selects elements based on the visible text.
     - Example: `cy.contains('Submit')` selects the element containing the text `Submit`.

---

1. **How do you handle asynchronous operations in Cypress?**
   - Cypress automatically waits for elements to be available and commands to complete.
   - Example: Instead of adding manual waits like `sleep()`, Cypress retries commands like `cy.get()` until the element appears.

Selectors and Element Interaction

1. **What types of selectors are supported by Cypress?**
   - **Types**:
     - CSS Selectors (default).
     - Text-based selectors using `cy.contains()`.
     - XPath (requires a plugin).
     - Aliases (`@aliasName` for reusable references).

---

1. **What is the Cypress Command Log, and how is it useful?**
   - **Definition**: A visual representation of all Cypress commands executed during a test run.
   - **Usage**:
     - Debugging: Inspect each step of a test to understand failures.
     - Time travel: Hover over commands to see the DOM state at each point.

---

1. **How do you interact with elements in Cypress?**
    - **Common Methods**:
        - `click()`: Simulates a mouse click.
        - `type('text')`: Types text into an input field.
        - `check()`: Checks a checkbox or radio button.
        - `select('Option')`: Selects an option from a dropdown.

---

1. **How do you handle dynamic elements in Cypress?**
    - Use robust selectors like data attributes (e.g., `[data-cy="submit"]`).

Add retries with assertions:

javascript
Copy code
cy.get('.dynamic-element').should('exist');

    -

---

1. **What is the difference between `cy.get()` and `cy.find()`?**
    - `cy.get()`: Starts a new query for elements.
    - `cy.find()`: Searches for elements within a parent element already found.

Advanced Features

1. **How do you handle dropdowns in Cypress?**

Use the `select()` method:

javascript
Copy code
cy.get('select').select('Option1');

    -

---

1. **How do you handle alerts and pop-ups in Cypress?**
    - Cypress automatically handles browser alerts.

Example:

javascript
Copy code
```
cy.on('window:alert', (alertText) => {
  expect(alertText).to.equal('This is an alert');
});
```

    -

---

1. **How do you switch between browser tabs or iframes in Cypress?**
    - **Tabs**: Cypress cannot directly handle multiple tabs but can verify link targets with `cy.request()`.
    - **Iframes**: Use `cy.frameLoaded()` and `cy.iframe()` with a plugin.

---

1. **What are Cypress aliases, and how are they used?**
    - **Definition**: Aliases allow storing references for elements, routes, or variables for reuse.

Example:

javascript
Copy code
```
cy.get('#username').as('userInput');
cy.get('@userInput').type('user123');
```

    -

---

1. **How do you stub and spy on network requests in Cypress?**
    - Use `cy.intercept()` to intercept or mock network requests.

Example:

javascript
Copy code
```
cy.intercept('GET', '/api/data', { fixture: 'data.json' }).as('getData');
cy.wait('@getData').its('response.statusCode').should('eq', 200);
```

-

Synchronization and Waits

1. **What is automatic waiting in Cypress?**
   - Cypress automatically waits for commands and assertions to complete, unlike Selenium where explicit waits are required.

---

1. **What is the difference between `cy.wait()` and `cy.intercept()`?**
   - `cy.wait()`: Delays execution for a specified time or waits for a stubbed network call.
   - `cy.intercept()`: Captures, modifies, or mocks network requests.

Handling Special Scenarios

1. **How do you test file uploads in Cypress?**
   - Use the `attachFile()` method from a plugin like `cypress-file-upload`.

---

1. **How do you handle CAPTCHA or two-factor authentication in Cypress?**
   - Mock responses or use test-specific keys.
   - Automating CAPTCHA is discouraged for ethical reasons.

---

1. **What are Cypress's capabilities for API testing?**

   - Cypress has built-in commands like `cy.request()` to send API requests and validate responses.

javascript
Copy code
cy.request('/api/data').its('status').should('eq', 200);

2.

Scenario-Based Questions

1. **How would you write a script to test a login page in Cypress?**