100 Must-Know Cypress Interview Q&A to Land That Offer!

Curated by : Lamhot Siagian LinkedIn

Chapter 1: Getting Started with Cypress

1. **What is Cypress? Answer:** Cypress is a next-generation, JavaScript-based end-to-end testing framework that runs directly in the browser, addressing common issues like flaky tests and synchronization by operating at the network and DOM level within the same run loop. (rahulshettyacademy.com)

   ```
   // Basic Cypress test skeleton
   describe('Getting Started', () => {
     it('loads the home page', () => {
       cy.visit('https://example.com');
     });
   });
   ```

2. **How is Cypress architecture different from Selenium? Answer:** Unlike Selenium—which sends commands over WebDriver to external browser processes—Cypress runs inside the browser process itself, giving direct access to the DOM, network layer, and native events, eliminating network lag and synchronization hassles. (rahulshettyacademy.com)

   ```
   // No extra driver setup needed
   cy.visit('https://example.com');
   ```

3. **Which languages and runtimes does Cypress support? Answer:** Cypress tests are written in JavaScript or TypeScript and run on Node.js; it does *not* support languages like Java or C#. (rahulshettyacademy.com)

   ```
   # Install Cypress via npm
   npm install cypress --save-dev
   ```

4. **What are the main features of Cypress? Answer:** Automatic waiting, time-travel debugging, real-time reloads, network traffic control, and built-in assertions make Cypress uniquely reliable and developer-friendly. (rahulshettyacademy.com)

   ```
   cy.get('button').should('be.visible').click();
   ```

5. **How do you install Cypress? Answer:** Run `npm install cypress --save-dev` or `yarn add cypress --dev`. (rahulshettyacademy.com)

   ```
   npm install cypress --save-dev
   ```

6. **How do you open the Cypress Test Runner? Answer:** Use `npx cypress open` to launch the interactive GUI, or `npx cypress run` for headless mode. (rahulshettyacademy.com)

   ```
   npx cypress open
   ```

7. **Where does Cypress expect your spec files by default? Answer:** In the `cypress/e2e` (or previously `cypress/integration`) folder. (rahulshettyacademy.com)

   ```
   /cypress
     /e2e
       example.spec.js
   ```

8. **What is the Cypress configuration file and how do you modify it? Answer:** `cypress.config.js` (or `cypress.json` pre-v10) holds project settings like `baseUrl`, `viewportWidth`, etc. You can edit it directly or override via CLI flags. (docs.cypress.io)

   ```
   // cypress.config.js
   module.exports = {
     e2e: {
       baseUrl: 'https://staging.example.com',
       viewportWidth: 1280,
       viewportHeight: 800,
     }
   };
   ```

9. **How can you run a single spec file from the CLI? Answer:**

   ```
   npx cypress run --spec "cypress/e2e/login.spec.js"
   ```

   (rahulshettyacademy.com)

10. **How do you run Cypress in headless mode? Answer:**

    ```
    npx cypress run --headless --browser chrome
    ```

    (rahulshettyacademy.com)

Chapter 2: Installing & Configuring Your Environment

1. **How do you set environment variables in Cypress? Answer:** Via `cypress.config.js` under the `env` key, a separate `cypress.env.json`, CLI flags (`--env key=value`), or OS env vars prefixed with `CYPRESS_`. (medium.com, docs.cypress.io)

   ```
   // cypress.config.js
   module.exports = {
     env: {
   ```

```
    apiUrl: 'https://api.example.com',
  }
};
```

2. **How do you access an environment variable in a test? Answer:**

   ```
   const url = Cypress.env('apiUrl');
   cy.request(url + '/users').its('status').should('eq', 200);
   ```

   (docs.cypress.io)

3. **What are `baseUrl` and `defaultCommandTimeout`? Answer:** `baseUrl` lets you omit full URLs in `cy.visit()`, while `defaultCommandTimeout` sets the retry time for commands/assertions. (docs.cypress.io)

   ```
   // cypress.config.js
   module.exports = {
     e2e: {
       baseUrl: 'https://app.example.com',
       defaultCommandTimeout: 10000,
     }
   };
   ```

4. **How do you configure video recording and screenshots? Answer:** In `cypress.config.js`:

   ```
   module.exports = {
     e2e: {
       video: true,
       screenshotsFolder: 'cypress/screenshots',
     }
   };
   ```

   (docs.cypress.io)

5. **How can you override configuration mid-test? Answer:** Use `Cypress.config()` in your spec:

   ```
   Cypress.config('defaultCommandTimeout', 20000);
   ```

   (docs.cypress.io)

6. **What's the difference between `cypress.env.json` and `cypress.config.js` `env`? Answer:** `cypress.env.json` is git-ignored by convention for secrets; its values override `config.js` env settings. (medium.com)

7. **How do you set per-test configuration? Answer:** Pass a config object to `it` or `describe`:

   ```
   it('runs with custom config', { env: { debug: true } }, () => {
     // test body
   });
   ```

(docs.cypress.io)

8. **How do you install a specific Cypress version? Answer:**

```
npm install cypress@12.17.4 --save-dev
```

(docs.cypress.io)

9. **How do you pin Chrome version in CI? Answer:** Use `--browser chrome:114` or set the `CYPRESS_BROWSER` env var. (docs.cypress.io)

10. **How do you configure a TS project with Cypress? Answer:**

    1. Add `tsconfig.json`
    2. Rename specs to `.ts`
    3. Install `@types/node` and `cypress`.

```
npm install --save-dev typescript @types/node
```

(docs.cypress.io)

Chapter 3: Your First Cypress Test

1. **What commands do you use to write your first test? Answer:** Use `describe`, `it`, and core commands like `cy.visit()`, `cy.get()`, `cy.contains()`. (rahulshettyacademy.com)

```
describe('My First Test', () => {
  it('finds content on page', () => {
    cy.visit('https://example.com');
    cy.contains('Welcome').should('be.visible');
  });
});
```

2. **How do you chain commands in Cypress? Answer:** Cypress auto-waits and retries, so you can chain:

```
cy.get('input').type('Hello').should('have.value', 'Hello');
```

(rahulshettyacademy.com)

3. **How can you debug a failing test? Answer:** Use `.debug()`, `cy.pause()`, or `cy.log()`. (rahulshettyacademy.com)

```
cy.get('button').debug().click();
```

4. **How do you skip or only run a test? Answer:**

```
it.skip('skipped test', () => { ... });
it.only('only this test', () => { ... });
```

(rahulshettyacademy.com)

5. **How do you retry flaky tests? Answer:** Configure retries in `cypress.config.js`:

   ```
   module.exports = { retries: { runMode: 2, openMode: 1 } };
   ```

   (docs.cypress.io)

6. **How can you add custom timeouts per command? Answer:**

   ```
   cy.get('.item', { timeout: 20000 }).should('be.visible');
   ```

   (rahulshettyacademy.com)

7. **How do you group tests into suites? Answer:** Use nested `describe` blocks:

   ```
   describe('User Flows', () => {
     describe('Login', () => {
       it('logs in', () => {...});
     });
   });
   ```

8. **How can you assert on page title? Answer:**

   ```
   cy.title().should('include', 'Dashboard');
   ```

   (rahulshettyacademy.com)

9. **How do you capture screenshots on failure? Answer:** Enable `screenshotOnRunFailure: true` in config. (docs.cypress.io)

10. **How do you write a data-driven test? Answer:** Iterate over fixtures:

    ```
    cy.fixture('users.json').each((user) => {
      cy.visit('/login');
      cy.get('#email').type(user.email);
      cy.get('#password').type(user.pass);
      cy.get('button').click();
      cy.contains('Welcome').should('exist');
    });
    ```

    (rahulshettyacademy.com)

Chapter 4: Finding & Interacting with Elements

1. **What selector strategies are supported? Answer:** CSS selectors, XPath (via plugin), content with `cy.contains()`, and data attributes. (rahulshettyacademy.com)

   ```
   cy.get('[data-cy=submit]').click();
   ```

2. **How do you click an element? Answer:**

```
cy.get('button').click();
```

(rahulshettyacademy.com)

3. **How can you force-click hidden elements? Answer:** Use `{ force: true }`:

```
cy.get('.hidden-btn').click({ force: true });
```

(rahulshettyacademy.com)

4. **How do you type into input fields? Answer:**

```
cy.get('input[name=email]').type('test@example.com');
```

(rahulshettyacademy.com)

5. **How can you select an option in a dropdown? Answer:**

```
cy.get('select').select('Option 2').should('have.value', '2');
```

6. **How do you handle checkboxes and radio buttons? Answer:**

```
cy.get('#agree').check().should('be.checked');
cy.get('[type=radio]').check('value1');
```

(rahulshettyacademy.com)

7. **How can you trigger mouse events (hover, drag/drop)? Answer:** Use `.trigger()`:

```
cy.get('.draggable')
  .trigger('mousedown', { which: 1, pageX: 100, pageY: 100 })
  .trigger('mousemove', { pageX: 200, pageY: 200 })
  .trigger('mouseup');
```

(rahulshettyacademy.com)

8. **How do you upload files? Answer:** Use the `cypress-file-upload` plugin:

```
cy.get('input[type=file]').attachFile('test.pdf');
```

9. **How do you handle elements in shadow DOM? Answer:** With the `cypress-shadow-dom` plugin:

```
cy.get('my-element').shadow().find('button').click();
```

(rahulshettyacademy.com)

10. **How do you ensure element existence before action? Answer:**

```
cy.get('.widget', { timeout: 10000 }).should('exist');
```

(rahulshettyacademy.com)

Chapter 5: Assertions & Testing Strategies

1. **What assertion styles does Cypress support? Answer:** BDD (`should`) and TDD (`expect`) via Chai, Sinon, and jQuery. (rahulshettyacademy.com)

   ```
   cy.get('h1').should('contain.text', 'Welcome');
   expect(true).to.be.true;
   ```

2. **How do you chain multiple assertions? Answer:**

   ```
   cy.get('.item')
     .should('be.visible')
     .and('have.length', 3)
     .and('contain.text', 'Item 1');
   ```

   (rahulshettyacademy.com)

3. **How can you assert on network responses? Answer:**

   ```
   cy.intercept('GET', '/api/data').as('getData');
   cy.visit('/');
   cy.wait('@getData').its('response.statusCode').should('eq', 200);
   ```

   (docs.cypress.io)

4. **How do you test error states? Answer:** Stub a 500 response:

   ```
   cy.intercept('GET', '/api/items', { statusCode: 500 }).as('getError');
   cy.visit('/items');
   cy.wait('@getError');
   cy.contains('Error loading items').should('be.visible');
   ```

5. **What is the difference between `.should()` and `.then()`? Answer:** `.should()` retries until success; `.then()` runs once when all previous commands succeed. (rahulshettyacademy.com)

6. **How do you verify cookies and localStorage? Answer:**

   ```
   cy.getCookie('sessionId').should('exist');
   cy.window().then((win) => {
     expect(win.localStorage.getItem('token')).to.eq('abc123');
   });
   ```

7. **How can you stub window alerts and confirms? Answer:**

   ```
   cy.on('window:alert', (txt) => expect(txt).to.contains('Are you sure?'));
   cy.get('button.delete').click();
   ```

8. **How do you assert element absence? Answer:**

   ```
   cy.get('.loading').should('not.exist');
   ```

9. **How can you verify page navigation? Answer:**

```
cy.url().should('include', '/dashboard');
cy.location('pathname').should('eq', '/dashboard');
```

10. **What is snapshot testing in Cypress? Answer:** Using the `cypress-image-snapshot` plugin to compare screenshots over time:

    ```
    cy.matchImageSnapshot('homepage');
    ```

Chapter 6: Network Control & API Testing

1. **What is `cy.intercept()`? Answer:** A powerful command to spy on, stub, and mock HTTP requests and responses in Cypress tests. (docs.cypress.io)

   ```
   cy.intercept('GET', '/users').as('getUsers');
   ```

2. **How do you mock a JSON response? Answer:**

   ```
   cy.intercept('GET', '/api/data', { fixture: 'data.json' }).as('mockData');
   ```

3. **How can you simulate a network error? Answer:**

   ```
   cy.intercept('GET', '/api/data', { forceNetworkError: true }).as('error');
   ```

4. **How do you wait for and assert on an intercepted call? Answer:**

   ```
   cy.wait('@getUsers').its('response.body').should('have.length', 5);
   ```

5. **How can you throttle network speed? Answer:**

   ```
   cy.intercept({ url: '/api/data', middleware: true }, (req) => {
     req.on('before:response', (res) => {
       res.setDelay(2000);
     });
   });
   ```

6. **What's the difference between `cy.route()` and `cy.intercept()`? Answer:** `cy.route()` is legacy (pre-v6); `cy.intercept()` replaces it with more flexible routing and stubbing. (medium.com)

7. **How do you spy on POST requests? Answer:**

   ```
   cy.intercept('POST', '/login').as('login');
   cy.get('button').click();
   cy.wait('@login').its('request.body').should('include', { user: 'abc' });
   ```

8. **How can you modify response headers? Answer:**

   ```
   cy.intercept('GET', '/api/data', (req) => {
     req.reply((res) => {
       res.headers['x-custom'] = '123';
   ```

```
  });
});
```

9. **How do you test file downloads? Answer:** Spy on the request and assert on response headers:

```
cy.intercept('GET', '/download').as('download');
cy.get('a.download').click();
cy.wait('@download').its('response.headers').its('content-type').should('eq', 'applicat
```

10. **How do you combine UI and API coverage? Answer:** Chain UI actions with API assertions:

```
cy.intercept('POST', '/items').as('addItem');
cy.get('#add').click();
cy.wait('@addItem').its('response.statusCode').should('eq', 201);
cy.contains('Item added').should('be.visible');
```

Chapter 7: Custom Commands & Utilities

1. **What are custom commands in Cypress? Answer:** User-defined commands to encapsulate reusable test steps and reduce duplication. (medium.com)

```
Cypress.Commands.add('login', (email, pwd) => {
  cy.visit('/login');
  cy.get('#email').type(email);
  cy.get('#password').type(pwd);
  cy.get('button').click();
});
```

2. **How do you overwrite an existing command? Answer:** Use `Cypress.Commands.overwrite(name, fn)`:

```
Cypress.Commands.overwrite('visit', (orig, url, options) => {
  // custom logic
  return orig(url, options);
});
```

(docs.cypress.io)

3. **How can you add multiple commands at once? Answer:**

```
Cypress.Commands.addAll({
  login(email, pwd) { ... },
  logout() { ... },
});
```

4. **Why use custom commands? Answer:** To improve readability, maintainability, and DRYness of your tests. (medium.com)

5. **How do you handle asynchronous setup in a command? Answer:** Return a Cypress chain:

```
Cypress.Commands.add('seedDatabase', () => {
  return cy.request('POST', '/api/seed');
});
```

6. **Where should you define custom commands? Answer:** In `cypress/support/commands.js` (or `.ts`). They're auto-loaded before specs. (docs.cypress.io)

7. **How do you type-safe custom commands in TypeScript? Answer:** Extend the `Cypress.Chainable` interface in `cypress/support/index.d.ts`:

```
declare namespace Cypress {
  interface Chainable {
    login(email: string, pwd: string): Chainable<void>;
  }
}
```

8. **How can you debug a custom command? Answer:** Use `Cypress.log()` inside the command or call `.debug()` when consuming it. (docs.cypress.io)

9. **How do you group common test utilities? Answer:** Create helper modules under `cypress/support/utils.js` and import in tests:

```
import { formatDate } from '../support/utils';
```

10. **How do you ensure commands have built-in retryability? Answer:** Return a jQuery element or chainable from the command so Cypress will retry automatically. (docs.cypress.io)

Chapter 8: Test Architecture & Design Patterns

1. **What is the Page Object Model (POM) in Cypress? Answer:** Encapsulate page structure and actions in classes/modules to keep tests DRY. (rahulshettyacademy.com)

```
// cypress/pages/LoginPage.js
class LoginPage {
  visit() { cy.visit('/login'); }
  fillForm(u, p) {
    cy.get('#email').type(u);
    cy.get('#password').type(p);
  }
  submit() { cy.get('button').click(); }
}
export default new LoginPage();
```

2. **How do you organize tests by feature? Answer:** Mirror your application's structure under `cypress/e2e/<feature>/…`. (rahulshettyacademy.com)
3. **What are test fixtures and why use them? Answer:** Static JSON files under `cypress/fixtures` for test data, kept separate from code. (rahulshettyacademy.com)

   ```
   cy.fixture('users').then((users) => { ... });
   ```

4. **How do you share state between tests? Answer:** Avoid shared state; use fixtures or custom commands to set up fresh data per test. (docs.cypress.io)
5. **What is a "test utility" vs. a "custom command"? Answer:** Utilities are plain JS functions; custom commands wrap Cypress commands for retryability. (docs.cypress.io)
6. **How do you manage test data? Answer:** Use fixtures, factories, or API calls in `beforeEach`, cleaning up with API teardown. (rahulshettyacademy.com)
7. **What's the role of `before`, `beforeEach`, `afterEach`, `after`? Answer:** Hooks for one-time or per-test setup/teardown.

   ```
   beforeEach(() => cy.login('u','p'));
   afterEach(() => cy.clearCookies());
   ```

8. **How can you parallelize Cypress suites? Answer:** In CI, split specs across containers and use Dashboard Service for load balancing. (rahulshettyacademy.com)
9. **What is test isolation and why is it important? Answer:** Ensuring each test runs in a fresh browser context (via `cy.clearCookies()`, `cy.visit()`), preventing flakiness. (rahulshettyacademy.com)
10. **How do you structure helpers for API vs. UI tests? Answer:** Separate modules under `cypress/support/api.js` and `cypress/support/ui.js` for clear boundaries. (rahulshettyacademy.com)

Chapter 9: CI/CD Integration & Reporting

1. **How do you run Cypress in GitHub Actions? Answer:** Use the official Cypress GitHub Action:

   ```
   - uses: cypress-io/github-action@v5
   ```

2. **How do you generate JUnit XML reports? Answer:** Install `mochawesome` and configure reporter in `cypress.config.js`:

```
reporter: 'junit',
reporterOptions: { mochaFile: 'results/junit-[hash].xml' }
```

3. **How can you upload artifacts to CI? Answer:** Save `cypress/screenshots` and `cypress/videos` as build artifacts in your CI config.
4. **How do you parallelize runs on CircleCI? Answer:** Split specs via `circleci tests split --split-by=timings`.
5. **How do you record runs to the Cypress Dashboard? Answer:**

   ```
   npx cypress run --record --key $CYPRESS_RECORD_KEY
   ```

6. **How can you retry on CI only? Answer:**

   ```
   // cypress.config.js
   retries: { runMode: 2, openMode: 0 }
   ```

7. **How do you set up environment-specific CI variables? Answer:** Configure `CYPRESS_envVar` in your CI provider's settings. (browserstack.com)
8. **How can you fail fast on CI? Answer:** Use the `--exit` and `--forbid-only` flags:

   ```
   npx cypress run --exit --forbid-only
   ```

9. **How do you integrate Cypress with Slack notifications? Answer:** Use a CI plugin or custom script to post results JSON to Slack API after the run.
10. **How do you visualize pass/fail trends over time? Answer:** Use the Cypress Dashboard's Analytics tab or integrate with Grafana via JSON exports. (rahulshettyacademy.com)

Chapter 10: Debugging, Best Practices & Ecosystem

1. **How do you debug network failures? Answer:** Use `cy.intercept()` with the `middleware: true` option and inspect the request/response object in `.then()`. (medium.com)

   ```
   cy.intercept({ url: '/api/*', middleware: true }, (req) => { req.continue((res) => { co
   ```

2. **How do you use Chrome DevTools with Cypress? Answer:** In headed mode, open DevTools via F12 in the Test Runner.
3. **What are common causes of flakiness and how to fix? Answer:** Timing issues (fix with retries/timeouts), shared state (clear cookies), and improper selectors (use data attributes). (rahulshettyacademy.com)

4. **How can you handle tests running too slowly? Answer:** Parallelize, disable video in CI, stub heavy network calls, and use `cy.session()` for auth caching. (docs.cypress.io)
5. **What plugins should you consider? Answer:**
   - `cypress-axe` (accessibility)
   - `cypress-image-snapshot` (visual diffing)
   - `cypress-file-upload` (file inputs) (docs.cypress.io)
6. **How do you keep tests maintainable? Answer:** Follow POM, use custom commands, keep fixtures small, and enforce linting on test code. (rahulshettyacademy.com)
7. **What linting tools work with Cypress? Answer:** ESLint with `eslint-plugin-cypress` to enforce best practices.

   ```
   npm install eslint-plugin-cypress --save-dev
   ```
8. **How do you measure test coverage? Answer:** Use `nyc/istanbul` with the `@cypress/code-coverage` plugin. (docs.cypress.io)
9. **What patterns help scale large codebases? Answer:**
   - Modular support files
   - Centralized selectors
   - Shared utilities (docs.cypress.io)
10. **Where can you find help and community plugins? Answer:**
    - Official docs: https://docs.cypress.io (docs.cypress.io)
    - Community plugins: https://www.npmjs.com/search?q=cypress

---

All Q&A sourced from Rahul Shetty Academy, Cypress Docs, Medium, and other trusted blogs.