UnCodemy

Cypress Notes PDF

Cypress is a JavaScript testing tool that runs in the browser and tests web apps end to end. These notes cover installing it, writing tests with cy commands, choosing stable selectors, assertions with automatic retries, stubbing network requests, fixtures, custom commands, the test runner, and running tests in CI, plus the main limits to know.

Key takeaways

- Cypress commands are queued and retried automatically until they pass or time out.
- Use stable selectors such as data attributes instead of CSS classes that change.
- cy.intercept lets you wait for, check or fake network requests.
- Tests should be independent; set up data through the API rather than the UI.
- Know the limits: browser support, multi-tab and cross-origin rules.

What is Cypress?

Cypress is an open-source JavaScript testing framework that runs tests inside the browser, with built-in retries, a visual runner and tools to stub network calls, mainly used for end-to-end testing of web applications.

What is Cypress and how is it different?

Cypress runs inside the browser alongside your app, which gives fast feedback, access to the page and automatic waiting. Tests are written in JavaScript or TypeScript with Mocha-style `describe` and `it` blocks and Chai-style assertions. It differs from Selenium, which controls the browser from outside through a driver, and has its own design for retries and time travel debugging in the runner.

How do I install and run it?

```
npm init -y
npm install cypress --save-dev
npx cypress open      # interactive runner
npx cypress run       # headless run, good for CI
```

Cypress creates a config file and a `cypress` folder with an `e2e` folder for tests, a `fixtures` folder and a `support` folder for shared commands.

What does a first test look like?

```
describe("login", () => {
  it("shows the dashboard after a valid login", () => {
    cy.visit("/login");
    cy.get("[data-cy=email]").type("demo@example.com");
    cy.get("[data-cy=password]").type("secret123");
    cy.get("[data-cy=submit]").click();
    cy.url().should("include", "/dashboard");
    cy.contains("Welcome").should("be.visible");
  });
});
```

Use a test account or made-up data and read passwords from environment variables, not from the test file.

How do commands and retries work?

Cypress commands such as `cy.get` and `cy.click` are added to a queue and run in order; they are asynchronous, so do not mix them with ordinary `await`. Queries and assertions retry until they pass or reach the timeout (four seconds by default). This removes most manual waits. Do not store the result of `cy.get` in a variable to use later; chain commands or use aliases with `.as()`.

How should I choose selectors?

Prefer dedicated attributes such as `data-cy` or `data-testid`. They are stable and show that the element is used by tests. Avoid long CSS paths and text that changes. `cy.contains` is useful for finding by visible text when the text is stable.

Which assertions are common?

```
cy.get(".item").should("have.length", 3);
cy.get("h1").should("have.text", "Orders");
cy.get("input").should("have.value", "abc");
cy.get("button").should("be.disabled");
cy.location("pathname").should("eq", "/cart");
```

Assertions chained with `should` retry automatically, which makes them robust against slow rendering.

How do I control network requests?

`cy.intercept` watches or replaces requests. Wait for a call to finish before checking the page, or fake a response to test errors and empty states.

```
cy.intercept("GET", "/api/products", { fixture: "products.json" }).as("getProducts");
cy.visit("/shop");
cy.wait("@getProducts");
cy.get("[data-cy=product]").should("have.length.greaterThan", 0);
```

What are fixtures and custom commands?

Fixtures are static data files in the `fixtures` folder, loaded with `cy.fixture` or used in intercepts. Custom commands add reusable steps through `Cypress.Commands.add`, such as `cy.login()`. Use them for repeated actions, and log in through the API where possible so that each test starts quickly.

How do I keep tests independent?

Each test should create the state it needs and not rely on another test. Clear cookies and storage between tests (Cypress does this by default), seed data through API calls or a test endpoint, and avoid long chains of UI steps just to reach a state.

Where does Cypress run in CI?

Run `npx cypress run` in a clean environment with a cached install, start the app first, record videos and screenshots on failure as build artifacts and use tags or folders to separate smoke tests from the full suite. Parallel runs need a service or manual splitting of spec files.

What are the limits to know?

Cypress supports Chromium-based browsers, Firefox and a WebKit option that may change; check the official documentation for the current list. It works with one browser tab at a time and has rules for visiting different origins in one test. Because tests run in the browser, tasks that need the operating system are done through plugins or tasks that run in Node.

What common mistakes should I avoid?

Using fixed waits (`cy.wait(5000)`), relying on CSS class names, writing long end-to-end tests that cover many features, sharing state between tests and trying to use async and await with cy commands.

What should I practise with Cypress?

Write tests against a public demo shop: sign in, search, add to cart and check totals. Then stub an API to return an empty list and an error, check both screens and run the suite headless from the command line. Add a custom login command that uses the API and compare run time with the UI login. These exercises cover most of what real projects need.

How does Cypress compare with other tools?

Cypress offers a fast, friendly runner and automatic retries for JavaScript teams, but it has different browser and multi-tab limits than some tools. Playwright and Selenium cover other needs. Choose by your application, team skills and browser requirements, and run a short proof of concept on a real flow before deciding.

Frequently asked questions

What is Cypress used for?

Cypress is a JavaScript tool for end-to-end and component testing of web applications, running tests inside the browser.

How does Cypress handle waiting?

Queries and assertions retry automatically until they pass or time out, so fixed waits are rarely needed.

How do I fake an API response in Cypress?

Use cy.intercept with a fixture or a static response to stub the request and test errors or empty states.

Can I use await with cy commands?

No. Cypress commands are queued and chained, so use chaining, aliases and then callbacks instead of await.

What are good selectors in Cypress?

Dedicated data attributes such as data-cy or data-testid, because they are stable and signal that tests use the element.

Sources and further reading

- Cypress documentation
- Cypress best practices

About this PDF

These PDFs are study material for learning and revision. They are not a certificate, a degree or a promise of a job, and technology changes, so check official documentation for the latest details. Spot an error? Write to info@uncodemy.com and the trainer team will review it.

More free resources and courses: uncodemy.com | info@uncodemy.com | +91 98183 66550