Cypress:

1. What is Cypress and how is it different from Selenium?

👉 Cypress is a modern JS-based end-to-end testing framework.

- **Runs in same browser context as app** (not outside like Selenium).
- **Faster & automatic waits** (no `sleep` needed).
- Built-in retry mechanism.
- Selenium supports multiple languages/browsers; Cypress mainly supports **JS/TS + Chrome-family/Firefox/WebKit**.

---

2. How does Cypress handle asynchronous behavior?

👉 Cypress commands are **asynchronous** but run inside a **command queue**. Cypress automatically waits for commands and assertions to pass before moving to the next. You use `.then()` to access resolved values.

---

3. What is the difference between `cy.get()` and `cy.find()`?

- `cy.get()` → finds elements from **DOM root**.
- `cy.find()` → finds elements within a **subject/parent element** (chained).

---

4. How do retries work in Cypress?

👉 Assertions (`should`, `expect`) and commands (`cy.get`) are **auto-retried** until timeout (default 4 sec). This reduces flakiness.

---

5. What's the difference between `.should()` and `.expect()` in Cypress?

- `should` → chain Cypress commands (`cy.get('h1').should('have.text', 'Abhishek')`) → retried.
- `expect` → regular Chai assertion inside `.then()` block → **not retried**.

---

6. What is the Cypress command queue?

👉 Cypress builds a queue of commands (`cy.visit`, `cy.get`, etc.). They don't execute immediately but in order. Promises are resolved behind the scenes, avoiding `async/await` confusion.

---

7. How do you handle flaky tests in Cypress?

- Use **auto-retry** assertions.
- Avoid arbitrary `cy.wait()`.
- Use test isolation (`testIsolation: true`).
- Use **network stubbing/mocking** when APIs are slow.
- Run retries in CI (`cypress.json → "retries": 2`).

---

8. What are Cypress fixtures and how do you use them?

👉 Fixtures are JSON/test data files stored in `cypress/fixtures/`.

Usage:

```
cy.fixture('user.json').then((user) => {
  cy.get('#username').type(user.name)
})
```

---

9. Difference between `cy.request()` and `cy.intercept()`?

- `cy.request()` → makes real API call (backend testing).
- `cy.intercept()` → stubs/mocks network calls (frontend isolation).

---

10. How do you handle file upload & download in Cypress?

- File upload → use **`cypress-file-upload` plugin** (`cy.get('input').attachFile('file.pdf')`).
- File download → verify file exists in `cypress/downloads/`.

---

11. How do you handle iframes in Cypress?

- Install `cypress-iframe` plugin.

```
cy.frameLoaded('#iframeId')
cy.iframe().find('button').click()
```

---

12. Can Cypress run tests in parallel?

👉 Yes, via **Cypress Dashboard service** (`--record --parallel` flags) or with Jenkins/GitHub Actions matrix builds. Helps reduce execution time.

---

13. What are Cypress custom commands?

👉 Reusable commands added in `cypress/support/commands.js`.

```
Cypress.Commands.add('login', (user, pass) => {
  cy.get('#username').type(user)
  cy.get('#password').type(pass)
  cy.get('button').click()
})
```

---

14. What are Cypress best practices for test isolation?

- Each test should **start clean** (new state).
- Use `beforeEach` for setup.
- Do not depend on test execution order.
- Use `cy.session()` for auth caching.

---

15. How to debug Cypress tests?

- `cy.log("message")`
- `cy.pause()` → stops test mid-run.
- `cy.debug()` → prints subject to console.
- Use Chrome DevTools (Cypress runs inside browser).

---

16. What reporting options are available in Cypress?

- Built-in **Mocha reports** (console).
- Plugins: **Mochawesome**, **Allure**, **JUnit**.
- Can integrate with Jenkins/GitHub Actions.

---

17. What are Cypress limitations?

- Single tab only (can’t switch multiple windows).
- Limited browser support (Chrome-family, Firefox, WebKit).
- No native mobile testing.
- Built only for JS/TS (not multi-language).

---

18. How do you do data-driven testing in Cypress?

- Use fixtures with arrays of data.

```
const users = require('../fixtures/users.json')
users.forEach(user => {
  it(`tests login for ${user.name}`, () => {
    cy.login(user.name, user.password)
  })
})
```

---

19. What's the difference between `cy.wrap()` and `cy.then()`?

- `cy.wrap(value)` → converts JS object/Promise into Cypress chainable.
- `cy.then()` → unwraps the subject for direct access.

---

20. How do you integrate Cypress in CI/CD (Jenkins/Docker)?

- Create **Dockerfile** with Cypress included.
- In Jenkins, run with `npx cypress run`.
- Use Allure/Mochawesome for reports.
- Use matrix builds for parallel execution.

==========

Top 20 Playwright (Python sync) Interview Questions & Answers

---

1. What is Playwright, and how is it different from Selenium?

- **Ans:** Playwright is a modern automation framework supporting Chromium, Firefox, and WebKit with a single API. Unlike Selenium, it supports auto-wait, built-in retries, faster execution, network mocking, multiple contexts, and headless execution by default.

---

2. How do you install Playwright in Python?

```
pip install playwright
playwright install
```

---

3. How do you launch a browser in Playwright (Python sync)?

```
from playwright.sync_api import sync_playwright

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)
    page = browser.new_page()
    page.goto("https://example.com")
    browser.close()
```

---

4. What are browser contexts?

- **Ans:** Contexts are like separate browser profiles. They allow running tests in isolation (cookies/storage/session). Faster than launching multiple browsers.

```
context = browser.new_context()
page = context.new_page()
```

---

5. Difference between `page.locator()` and `page.query_selector()`?

- `page.locator()` → **recommended**, auto-retry until element appears, handles multiple elements.
- `page.query_selector()` → returns immediately, no auto-retry.

---

6. How do you handle waits in Playwright?

- **Implicit waits:** Auto-waiting (locator actions wait until element is ready).
- **Explicit waits:** `page.wait_for_selector()`, `locator.wait_for()`.
- Example:

```
page.locator("#login").click()  # auto waits until clickable
```

---

7. How to handle alerts/popups?

```
page.once("dialog", lambda dialog: dialog.accept())
page.click("#alertBtn")
```

---

8. How to take a screenshot?

```
page.screenshot(path="screenshot.png")
```

---

9. How to handle multiple tabs/windows?

```
with page.expect_popup() as popup_info:
```

```
    page.click("#openTab")
new_page = popup_info.value
```

---

10. How to upload and download files?

- **Upload:**

```
page.set_input_files("input[type=file]", "file.txt")
```

- **Download:**

```
with page.expect_download() as download_info:
    page.click("#downloadBtn")
download = download_info.value
download.save_as("myfile.zip")
```

---

11. How to perform API testing with Playwright?

```
request_context = p.request.new_context()
response = request_context.get("https://api.github.com/users/octocat")
print(response.json())
```

---

12. How to mock network requests?

```
def handle_route(route):
    route.fulfill(status=200, body='{"message": "mocked!"}')

page.route("**/api/data", handle_route)
page.goto("https://example.com")
```

---

13. How do you assert in Playwright (Python)?

```
from playwright.sync_api import expect

expect(page.locator("#welcome")).to_have_text("Welcome")
expect(page).to_have_title("Login Page")
```

---

14. How do you handle authentication (cookies, storage)?

```
context = browser.new_context(storage_state="auth.json")
page = context.new_page()
```

- Or save login:

```
context.storage_state(path="auth.json")
```

---

15. How to record tests in Playwright?

```
playwright codegen https://example.com
```

→ Opens browser, records steps, and generates code.

---

16. How to run Playwright tests in parallel?

- Using **pytest + pytest-xdist**:

```
pytest -n 4
```

---

17. How do you handle dynamic elements?

- Use flexible locators:

```
page.locator("text=Login").click()
page.locator("button:has-text('Submit')").click()
```

---

18. How to debug tests in Playwright?

- `headless=False` to see browser.
- Add:

```
page.pause()  # Opens Playwright inspector
```

---

19. How to capture console logs or network logs?

```
page.on("console", lambda msg: print(msg.text))
page.on("request", lambda req: print(">>", req.url))
page.on("response", lambda res: print("<<", res.url))
```

---

20. How to integrate Playwright with CI/CD (Jenkins/Docker)?

- Add dependencies in Dockerfile.
- Run with pytest:

```
pytest --alluredir=allure-results
```

- Jenkins → publish allure results.