



Core Concepts & Commands

Basic Structure

A Cypress test file typically includes `describe` blocks (test suites) and `it` blocks (individual tests).

```
describe('My First Test', () => {
  it('Visits the Kitchen Sink', () => {
    // Test steps go here
  })
})
```

Visiting a Page

```
cy.visit(url) // Navigates to the specified URL.
cy.visit('https://example.cypress.io')
```

Finding Elements

```
cy.get(selector) // Gets one or more DOM elements by selector.
cy.get('.my-element')
cy.get('#my-input')
cy.get('button[type="submit"]')
```

```
cy.contains(content) // Gets elements that contain the specified text.
cy.contains('Submit')
cy.contains('a', 'Click me') // finds <a> tags containing 'Click me'
```

Advanced Commands & Concepts

Aliases

```
cy.as(alias) // Creates an alias for a DOM element that can be reused later.
cy.get('button').as('submitButton')
cy.get('@submitButton').click()
```

Waiting

```
cy.wait(time) // Pauses execution for a specified amount of time (in milliseconds).
cy.wait(2000) // Wait for 2 seconds
```

```
cy.wait('@alias') // Waits for a specific route to be hit (useful for API testing).
cy.intercept('GET', '/api/data').as('getData')
cy.visit('/page-that-fetches-data')
cy.wait('@getData')
```

Debugging Techniques

Interacting with Elements

```
.click() // Clicks a DOM element.
cy.get('button').click()
```

```
.type(text) // Types into a DOM element.
cy.get('input').type('Hello, Cypress!')
```

```
.clear() // Clears the value of an input or textarea.
cy.get('input').clear()
```

```
.check() / .unchecked() // Checks or unchecks a checkbox or radio button.
cy.get('input[type="checkbox"]').check()
cy.get('input[type="radio"]').unchecked()
```

```
.select(value) // Selects an option in a <select> element.
cy.get('select').select('Option 2')
```

Working with Forms

Cypress simplifies form interactions. Use `.type()`, `.check()`, `.select()` as shown before, and then submit the form.

```
cy.get('#name').type('John Doe')
cy.get('#email').type('john.doe@example.com')
cy.get('form').submit()
```

Cypress Studio

Cypress Studio allows you to record interactions and generate Cypress tests. Enable it in `cypress.config.js`:

```
module.exports = {
  e2e: {
    experimentalStudio: true,
  },
}
```

Then, right-click in the Cypress runner to start recording.

Assertions

`.should(matcher, value)` Asserts that a DOM element has the expected state or value.

```
cy.get('input').should('have.value', 'Hello')
cy.get('button').should('be.visible')
cy.get('.message').should('contain', 'Success!')
```

`.expect(value).to.be.equal(expected)` Uses Chai's `expect` syntax for more complex assertions.

```
cy.get('input').then(($input) => {
  expect($input.val()).to.equal('Expected Value')
})
```

Custom Commands

You can create custom commands to encapsulate reusable logic. Add them to

`cypress/support/commands.js`:

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

Then use them in your tests:

```
cy.login('testuser', 'password123')
```

Using Fixtures

Fixtures are used to load external data. Place your JSON data in the `cypress/fixtures` directory.

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

Using the Cypress Runner

The Cypress Runner provides a visual interface for running and debugging tests. It allows you to:

- Step through each command.
- Inspect the DOM at each step.
- View network requests.
- Time travel to previous states.

Debugging Commands

`.debug()` Inserts a `debugger` statement in your code, pausing execution and opening the browser's developer tools at that point.

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

`.pause()` Pauses the test execution, allowing you to inspect the current state in the Cypress Runner.

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

`.then(console.log)` Log the output to the console, for debugging purposes.

```
cy.get('button').then(console.log).click()
```

Configuration

cypress.config.js

The `cypress.config.js` file is used to configure Cypress. Common options include:

- `baseUrl`: The base URL for your application.
- `viewportWidth`: The width of the viewport.
- `viewportHeight`: The height of the viewport.
- `defaultCommandTimeout`: The default timeout for Cypress commands.

```
const { defineConfig } = require('cypress')
```

```
module.exports = defineConfig({
  e2e: {
    baseUrl: 'http://localhost:3000',
    viewportWidth: 1280,
    viewportHeight: 720,
    defaultCommandTimeout: 4000,
  },
})
```

Console Logging

Use `console.log()`, `console.warn()`, and `console.error()` to log information to the browser's console.

```
cy.get('button').then($button => {
  console.log('Button text:', $button.text())
})
```

Network Request Logging

Use the Cypress Runner to inspect network requests. You can see the request headers, body, and response.

For more advanced network request debugging, use `cy.intercept()` and `cy.wait()` to monitor API calls.

Common Errors

- **Timeouts:** Cypress has default timeouts. Increase them using the `timeout` option if needed.
`cy.get('element', { timeout: 10000 })`
- **Detached DOM:** Ensure elements are still attached to the DOM when interacting with them.
- **Incorrect Selectors:** Verify that your selectors are correctly targeting the desired elements.