ACADE MIND

What is cypress ?

ACADE MIND

What Is Cypress?

cypress

↓

Test Automation Tool & Framework

(for websites)

Use Cypress For Automated Testing

This Course

End-to-End (E2E) Tests

Test complete application flows

e.g., user authentication flow

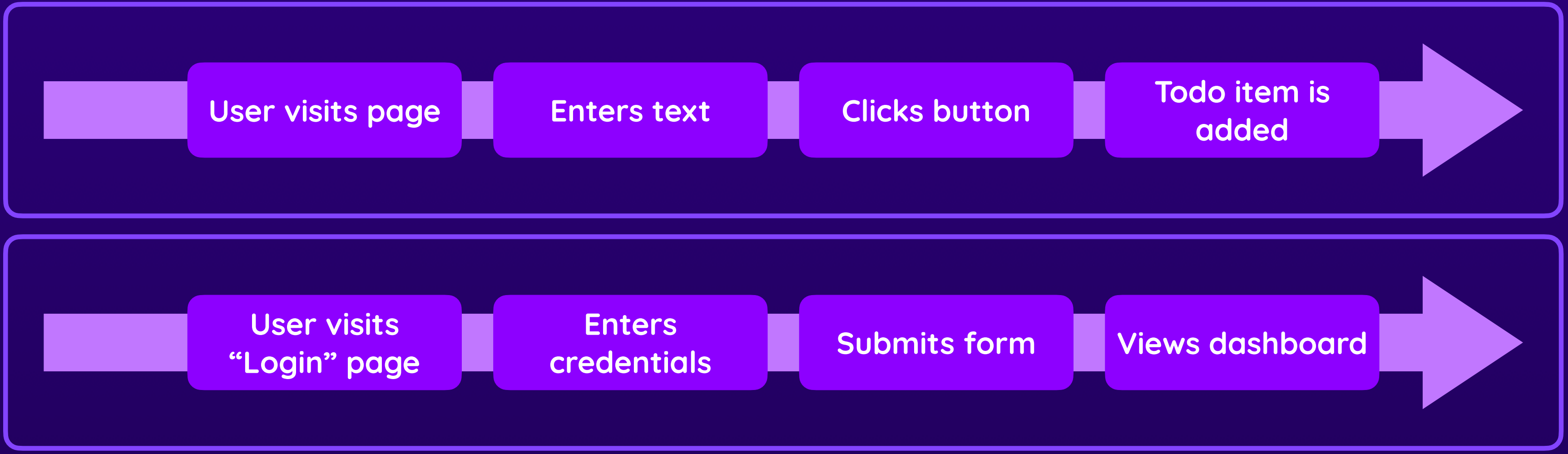
Component Tests

Test individual UI elements

e.g., a modal overlay component

What Is End-to-End (E2E) Testing?

Test application workflows from end to end



E2E vs Unit Testing

Unit Testing	E2E Testing
Test small app building blocks	Test complete application workflows
e.g., an individual function	e.g., login flow
Ensures correct functionality of individual units	Ensures correct functionality of core app features & processes
Does not guarantee functionality of overall system	Does not necessarily cover all building blocks of an app

About This Course

This is a “Getting Started” course!

No prior Cypress knowledge is required

Example-based explanation of core concepts

Common problems & solutions are shown

How To Get The Most Out Of This Course

Watch the Videos

At your pace: Use the video player controls

On-Demand: Repeat videos & sections as needed

Code Along & Practice

Pause & try things on your own

Practice what you learned (also in your own projects)

Use attached slides & code snapshots

Help Each Other

Ask & answer in the Q&A section

Join our amazing Discord community!

ACADE MIND

Fundamentals & Basics

How to write E2E tests with Cypress

- Finding Page Elements
- Simulating User Interaction
- Writing Assertions & Evaluating Tests

Module Summary

Setup & Adding Tests

```
npm install cypress
npx cypress open
```

Store tests (`it()`) in suites (`describe()`)

Selecting Elements

Select elements via CSS with `get()` + `find()`

Select by text via `contains()`

Adding Steps / Commands

Use the `cy` object to define the executable steps

Commands / queries can be chained

Simulating User Interaction

Use actions like `click()` or `type()`

Expectations / Assertions

Many queries have built-in assertions (e.g., `get()`)

Add explicit assertions via `should()`

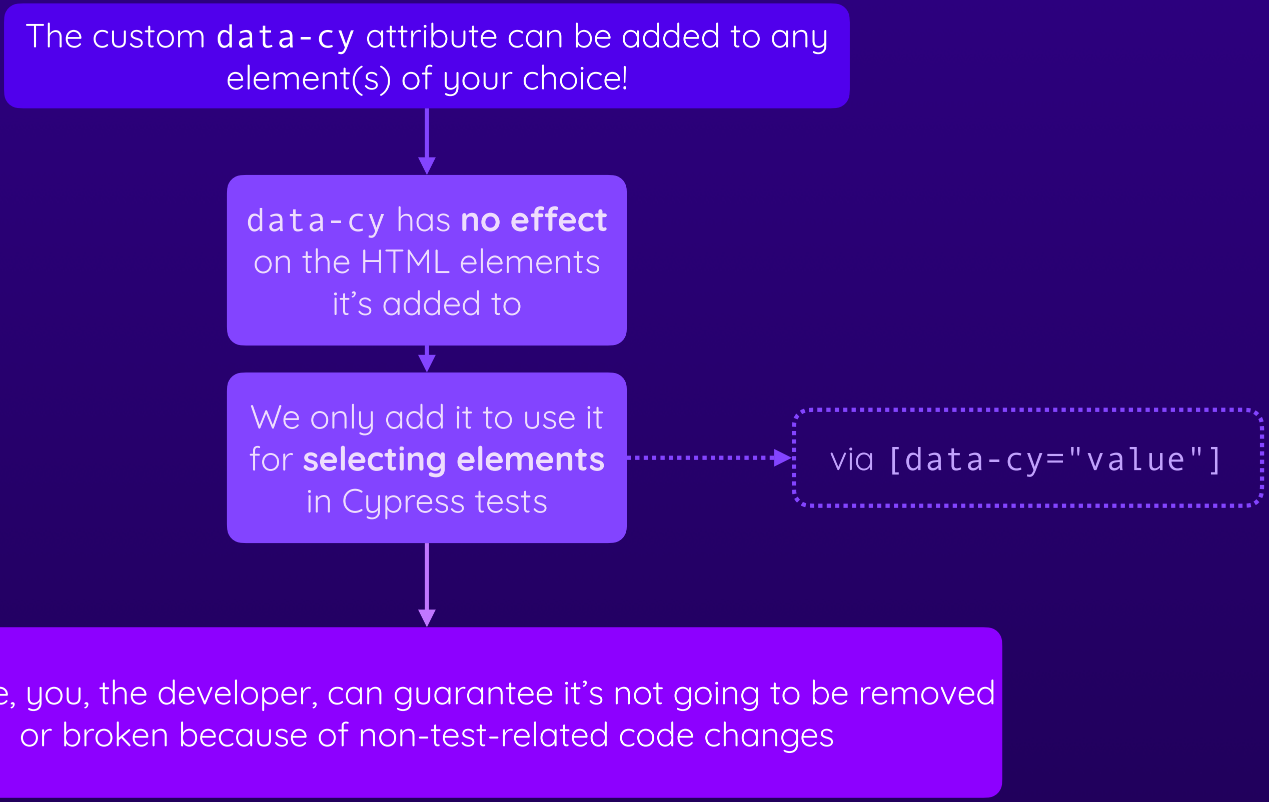
Add as many assertions as needed to test different flow states

Deep Dive: Select, Act, Assert

A closer look at element selection, actions & assertions

- Select & Use Elements Efficiently
- More Actions & Testing Page Navigations
- More on Assertions & should()

Prefer data-cy Selectors



The custom `data-cy` attribute can be added to any element(s) of your choice!

`data-cy` has **no effect** on the HTML elements it's added to

We only add it to use it for **selecting elements** in Cypress tests

via `[data-cy="value"]`

Therefore, you, the developer, can guarantee it's not going to be removed or broken because of non-test-related code changes

Selecting with data-cy

```
get('data-cy="my-element"')
```

```
<p>Hello world!</p>
<p data-cy="my-element">Selected</p>
```

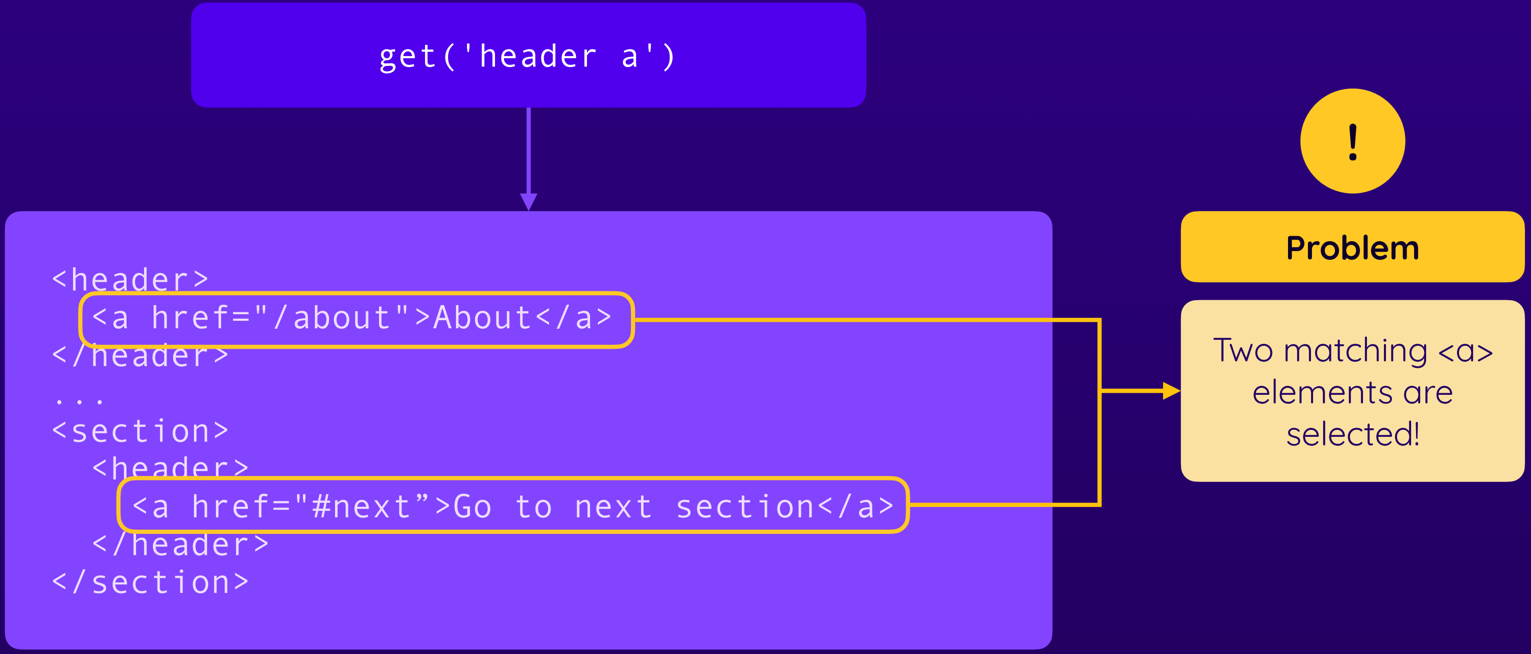
Only second paragraph is selected

```
<li data-cy="my-element">Item 1</li>
<li data-cy="my-element">Item 2</li>
<li data-cy="my-element">Item 3</li>
```

All three items are selected

Selections are stable, even as elements are moved around or CSS classes or element IDs are changed

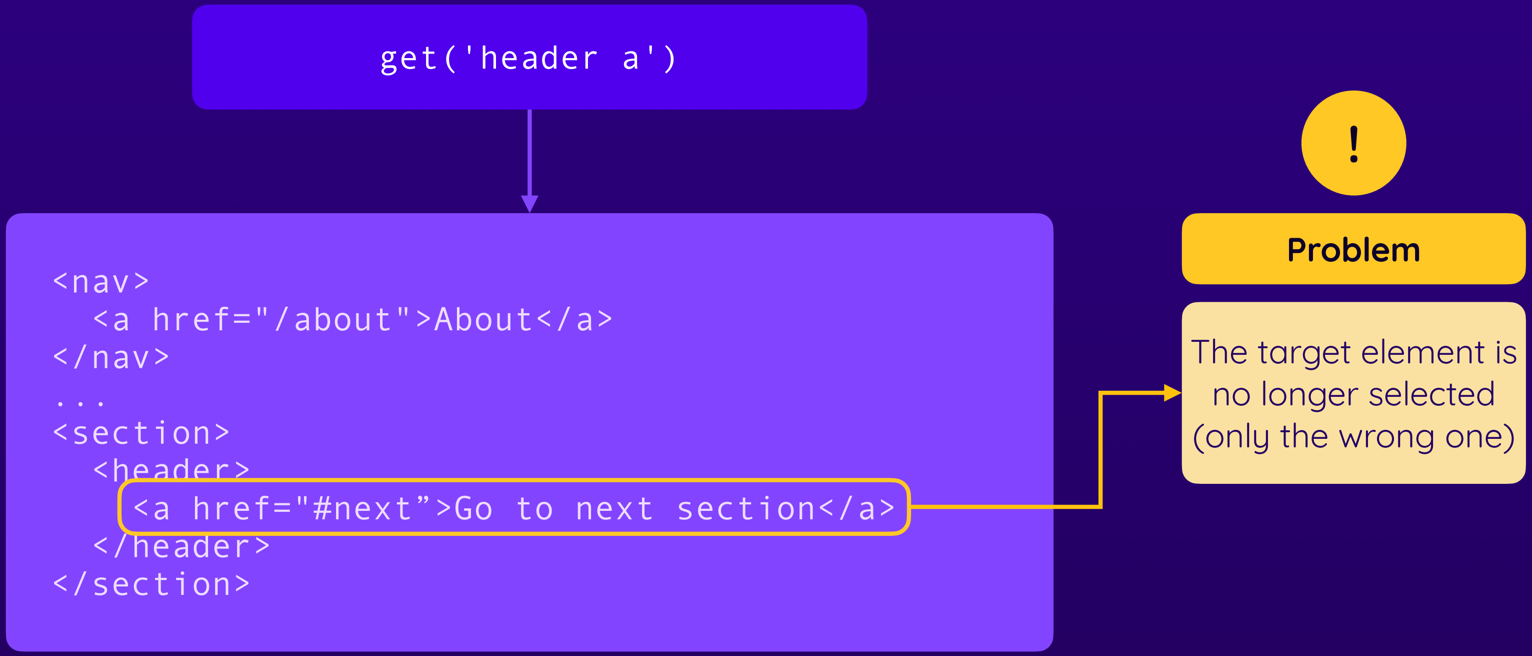
Dangerous Selectors



```
get('header a')
```

```
<header>
  <a href="/about">About</a>
</header>
...
<section>
  <header>
    <a href="#next">Go to next section</a>
  </header>
</section>
```

Dangerous Selectors



```
get('header a')
```

```
<nav>
  <a href="/about">About</a>
</nav>
...
<section>
  <header>
    <a href="#next”>Go to next section</a>
  </header>
</section>
```

!

Problem

The target element is no longer selected (only the wrong one)

Best Practice: Prefer data-cy

Prefer data-cy to avoid unwanted test failures because of DOM changes!

Module Summary

Selecting Elements

Prefer the `data-cy` attribute selector

It's less error-prone than other selectors

Use Aliases

Re-use query results via aliases

Create & use aliases via `as('name')` & `'@name'`

Get Element Access

Use `then()` for more direct element access

Different Assertion Approaches

`should()` vs `expect()`

Some `should()`s yield new subjects

Test Organization & Configuration

Being Efficient

- Configuring Tests & Timeouts
- Sharing Logic & Setup Steps Across Tests
- Custom Commands & Queries

Commands vs Queries

Commands

Re-usable "shortcuts" for more complex command chains

e.g., `cy.submitForm()` could be a custom command that finds the submit button in a form and clicks it

Queries

Synchronous, chainable, retriable commands

e.g., `cy.getById('abc')` could be a custom query that finds elements with `data-cy="abc"`

Executing Tasks

Tasks that should run outside of the browser

↓

Examples: Empty or delete a file, seed a database

ACADE MIND

Module Summary

Cypress Configuration

Global & local (test- or suite-specific)

e.g., set timeout values, browsers, baseUrl & more

Hooks

`before()`, `beforeEach()`

Test preparation or cleanup

Custom Commands & Queries

Outsource shared logic & command combinations

Don't overuse these features!

Tasks

Allow you to run code outside of the browser

Example: Seed database, store data in files, …

Spies, Stubs & Fixtures

Adjusting Testing Conditions

- Understanding Spies, Stubs & Fixtures
- Using Spies, Stubs & Fixtures
- Manipulating the Clock

Spies & Stubs

Spy	Stub
A listener that's attached to a function / method	A replacement for an existing function / method
Used for evaluating / asserting function calls	Used for evaluating & controlling function calls
Does **not** change or replace the function!	Does **replace** the function!

Only Test Your Application

What should your tests evaluate?

- ✓ Your Application
- ✗ Browser APIs
- ✗ 3rd Party APIs & Libraries

Module Summary

Stubs & Spies

Stubs: Replace existing methods

Spies: Add listeners to existing methods

Fixtures

Store dummy testing data in central place

Access via `fixture()` and use in your tests

Manipulating the Clock

Use `cy.clock()` to manipulate the clock

Then use `cy.tick()` to advance time

Network Requests, Databases & Auth

Dealing with the Tricky Parts

- Handling HTTP Requests in E2E Tests
- Using a Testing Database
- Simulating Authentication Flows

ACADE MIND

Dealing with Network Requests

Allow

Let the website do its requests

Potential problem: Database is hit with test data

Solution: Use a separate testing database

Intercept

Intercept + spy: Request passes & you can spy on it

Intercept + stub: Request is blocked & stub response is used

Trigger Manually

Test API endpoints from inside your tests

Ideal for API testing or for decoupling frontend & backend

Module Summary

Network Requests

Can be intercepted (and blocked)

Manually trigger requests for API testing

Test Database

Should be used when hitting the database

Ensures test isolation & avoids breaking live data

Authentication

Nothing special in general

Custom commands simplify your auth-dependent tests

You Did It!

You now have a solid understanding of the core Cypress concepts

- Use Cypress in your own projects
- Read the official docs
- Explore the Cypress "Real World App"