Cypress tutorials from scratch

What is Cypress?
Cypress is a next generation front end Automation testing tool built for the modern web applications

How Cypress is Unique from other Automation tools?

- Cypress automatically waits for commands and assertions before moving on. No more async hell.
- Ability to test Edge Testcases by Mocking the server response
- Cypress takes snapshots as your tests run. We can hover over each commands in the Command Log to see exactly what happened at each step.
- Because of its Architectural design, Cypress delivers fast, consistent and reliable test execution compared to other Automation tools
- View videos of your entire tests execution when run from the Cypress Dashboard.

****
Cypress built on Node.js and comes packaged as an npm module,
As it is built on Node.js, It uses JavaScript for writing tests. But 90% of coding can be done using Cypress inbuilt commands which are easy to understand.
Cypress also bundles with jQuery and inherits many of jQuery methods for UI components Identification

Cypress Architecture

Most testing tools (like Selenium) operate by running outside of the browser and executing remote commands across the network. But Cypress engine directly operates inside the browser. In other words, It is the browser that is executing your test code
This enables Cypress to listen and modify the browser behavior at run time by manipulating DOM and altering Network requests and responses on the fly
Cypress open doors to New Kind of testing with Having ultimate control over your application (front and back)

Cypress Browser Support:
Chrome
Electron. -> light weight browser of chromium family
Firefox & IE

Cypress Components:
Test Runner

Dash Board Service

Course Outcome:
By end of this course, You should be able to Automate any Web App using Cypress
You will understand how Cypress is Unique to build Non Flaky Stable Automation tests with the help of jQuery
You can mock network requests and responses with Cypress
Ability to Design Cypress framework from scratch with all the Testing standards
Integrate Cypress Test Framework to Jenkins for CI/CD

Course Prerequisites:

None for 90% of lectures.
Basics of API knowledge when dealing with API mocking topics. (10% lectures)
JavaScript Basics are taught in parallel when required with Cypress concepts

Topics Covered:

1. JavaScript Fundamentals
2. Web Automation with Cypress
3. API Automation with Cypress
4. Build Cypress Frameworks with Mocha and Cucumber
5. Intercepting Network responses & Browsers with Cypress
6. Database Testing with Cypress
7. Single Sign On & Accessibility Automation with Cypress

Cypress Step by Step Installation:

What is Node.js?
Node.js is an open source, cross-platform, back-end JavaScript run-time environment that runs on the V8 engine and executes JavaScript code outside a web browser.

Download Node.js

Download Visual Studio Code - (best editor for javascript)

Create new project with package.json
A **package.json** is a JSON file that exists at the root of a JavaScript/ Node project. It holds metadata relevant to the project and It is used for managing the project's dependencies.

Install Cypress

>npm install

Install Cypress via npm
>cd /your/project/path
>npm install cypress —save-dev

Npm - node package manager
Npm - Nuclear package magnet is node/ npm repository for dependencies

To create **package.json** file inside the project execute below command

```
>npm –i init
```

To resolve errors in VS Code:- our cache folder contains root-owned files, due to a bug in
sudo chown -R 501:20 "/Users/krishnabros/.npm"

To install latest version from npm:

```
>npm install cypress ––save–dev
```

You can now open Cypress by running: (CypressAutomation window launches on running below command)
In order to open the Cypress tests from command line:

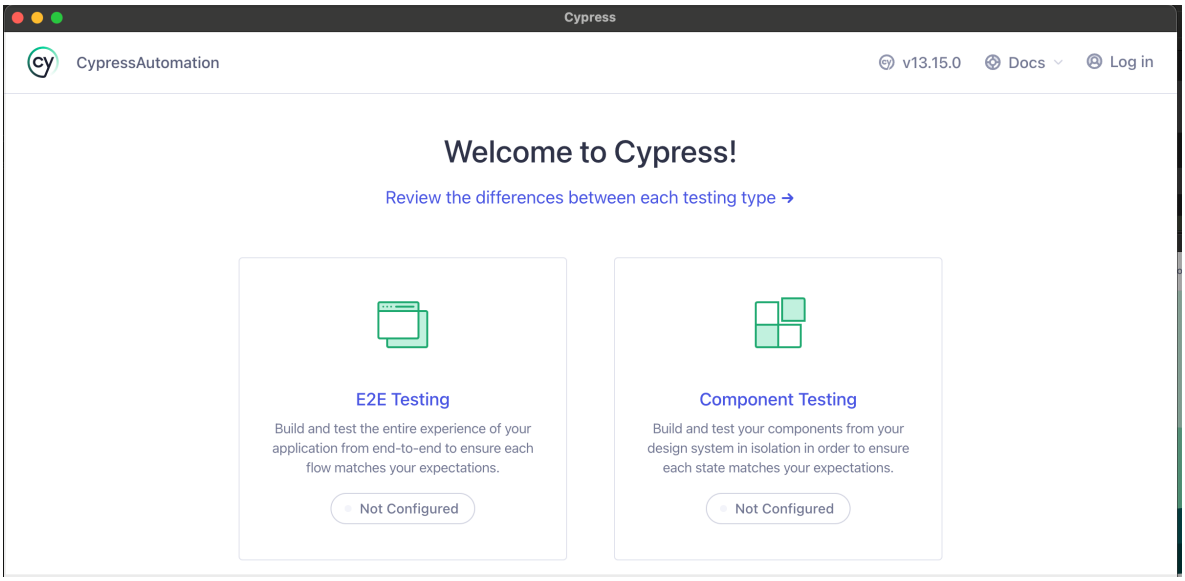
```
>node_modules/.bin/cypress open
```

OR

```
>npx cypress open
```

Add below lines of code in cypress package.json file and below command to launch cypressAutomation window:

```
"scripts": {
   "cy:open": "cypress open”,
    "cy:run": "cypress run"
 }

 npm run cy:open
```

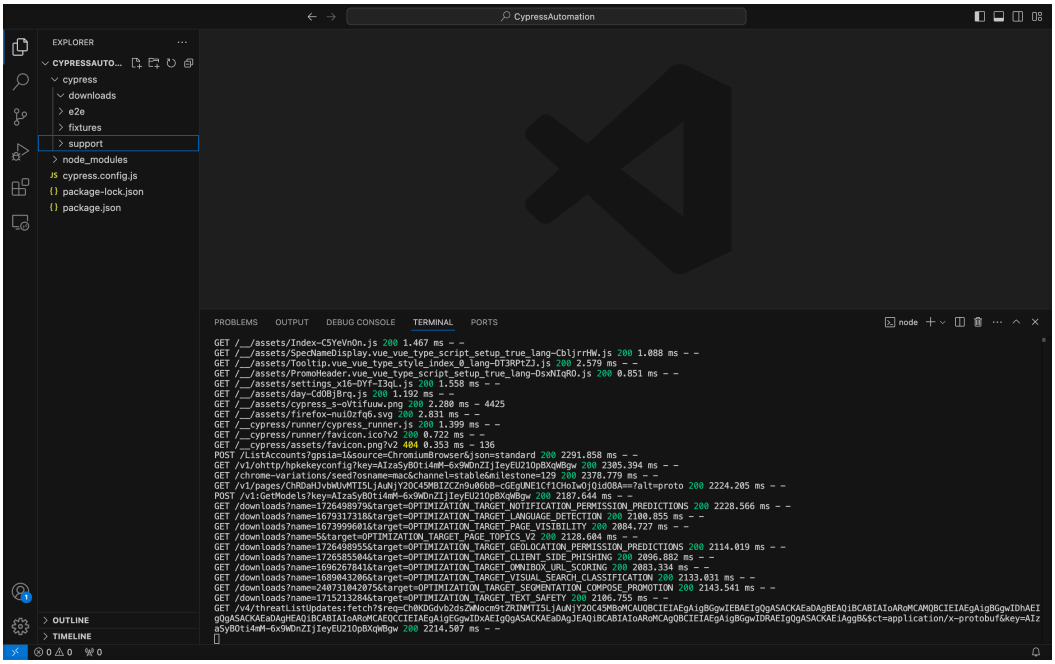


On selecting E2E Testing in cypressAutomation window, cypress folder & cypress.config.js file automatically added into our VS code cypress project

All 4 files are added automatically on choosing E2E Testing from cypressAutomation window.

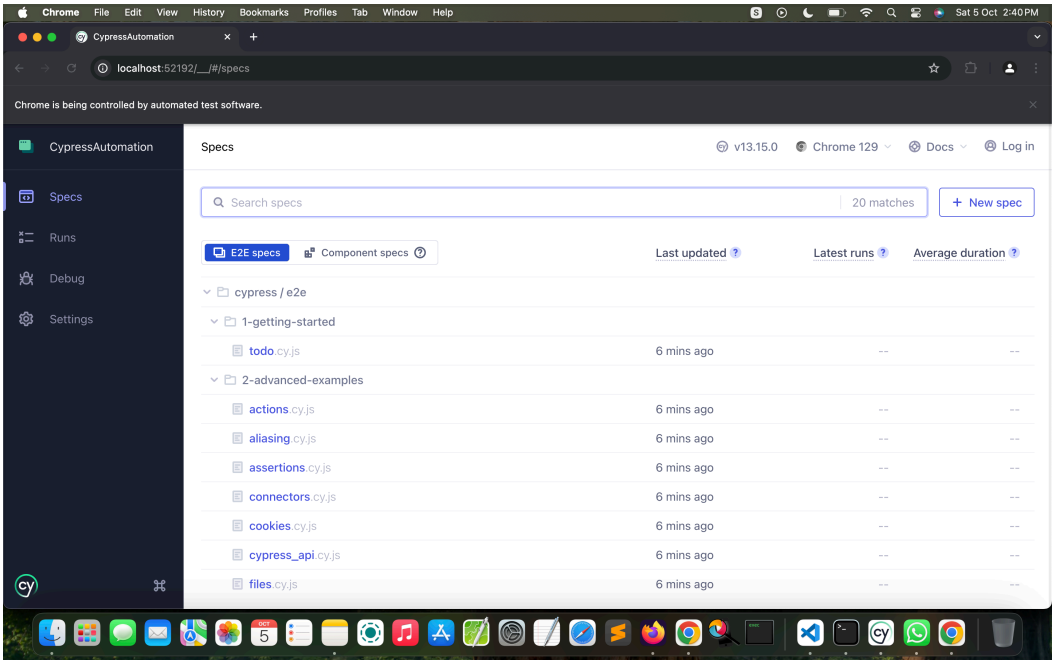
- cypress.config.js
- Cypress/support/e2e.js
- Cypress/support/command.js
- Cypress/fixtures/example.json

-cypress/download
-cypress/e2e
These 2 folders also created automatically after cypressAutomation window setup



Cypress TestRunner:

Below is the Test Runner created in cypress (E2E specs) :



Spec:

In JavaScript terminology, any test case is called as Spec file (or test file)
In java terminology, any test case we call as test file or .java file

Test / Spec files located in cypress project structure:

Cypress/Integration

Re-usable commands placed under:

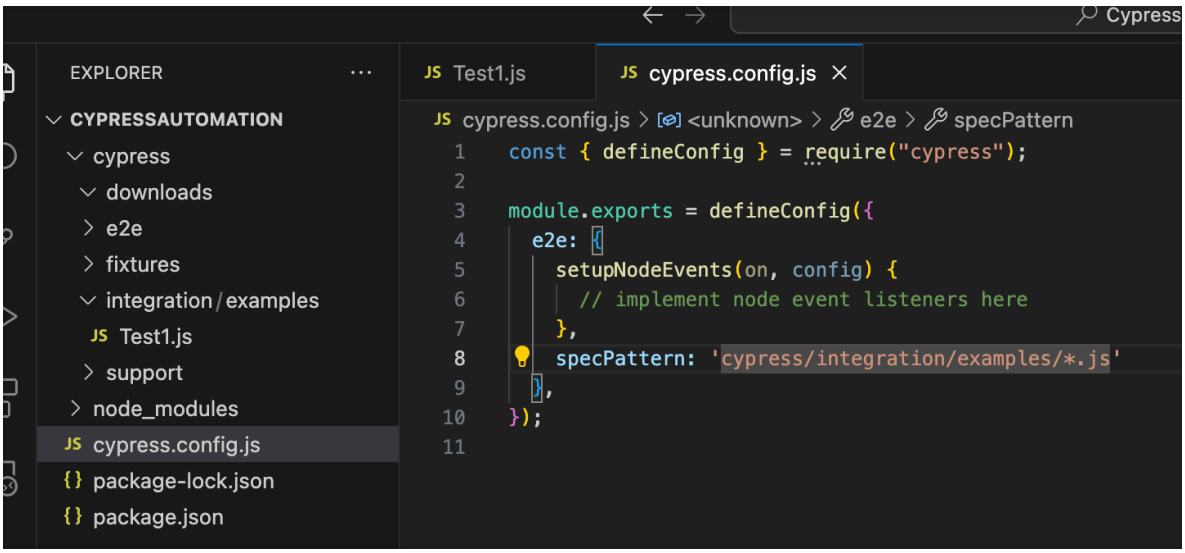
Support/command.js

defaultCommandTimeout time in milliseconds

4000

Where we override default configuration settings?

Cypress.config.js



What are *describe*, *it*, and *expect* ?
All of these functions come from Bundled Libraries that Cypress bakes in.

- **describe** and it come from Mocha
- **expect** comes from Chai

Cypress builds on these popular tools and frameworks that you *hopefully* already have some familiarity and knowledge of. If not, that's okay too.

In general terms,
`describe block  -> test suite`
`it block  -> test case`

Syntax:

```
describe('My First Test Suite', () => {
   it('My First Test Case', () => {
      //test steps
   }
   it('My Second Test Case', () => {
      //test steps
   })
})
```

Cy:
var cy: Cypress.cy & CyEventEmitter
Global variables cy added by Cypress with all API commands.

In order to run cypress tests from command line:

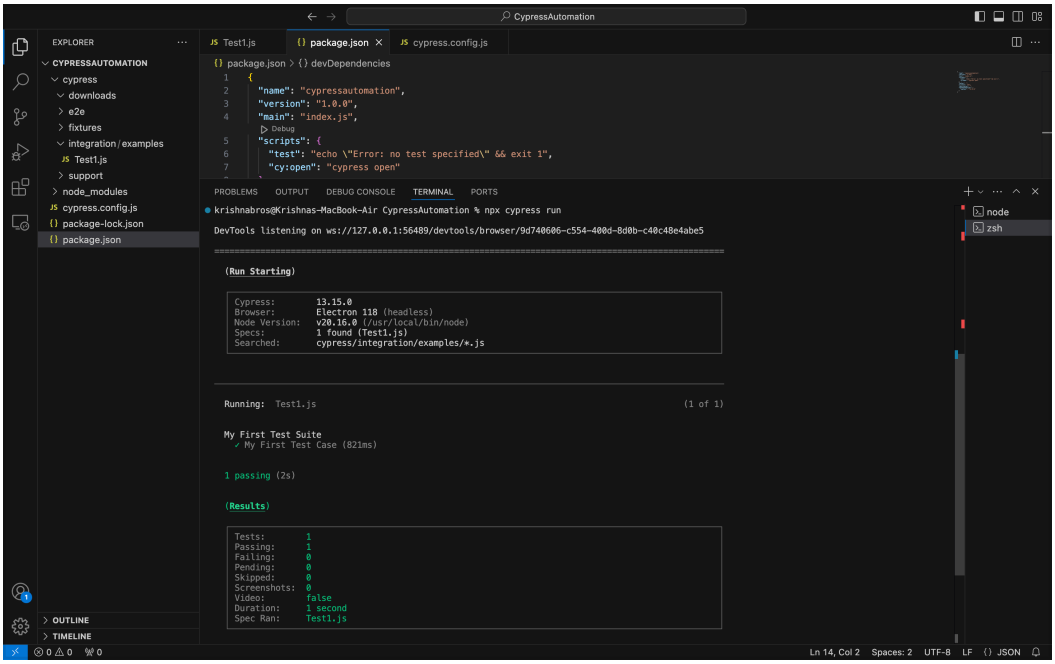
```
>npx cypress run
```

or

```
>node_modules/.bin/cypress run
```

```
"scripts": {
   "cy:open": "cypress open",
    "cy:run": "cypress run"
 }
```

```
npm run cy:run
```



In order to run cypress specific test from command line:

```
>npx cypress run --spec "cypress/integration/examples/my-spec.js"
```

Headless mode execution from CLI:

```
>npx cypress run
```

or

```
>node_modules/.bin/cypress run
```

Or

Add below lines of code in cypress package.json file and below command to launch cypressAutomation window:

```
"scripts": {
   "cy:open": "cypress open",
    "cy:run": "cypress run"
 }
```

```
>npm run cy:run
```

Always through command line, cypress tests executes in **headless mode.** (Without browser launch)
Default browser in which cypress test executes is in **Electron browser**

Headed mode execution from CLI(by default Electron browser used for cypress test execution):

```
>npx cypress run --headed
```

or

```
>node_modules/.bin/cypress run --headed
```

Or

Add below lines of code in cypress package.json file and below command to launch cypressAutomation window:

```
"scripts": {
   "cy:open": "cypress open —headed”,
    "cy:run": "cypress run —headed”
 }
```

```
>npm run cy:run
```

If we want to run cypress tests in Chrome browser through CLI:

```
>node_modules/.bin/cypress run --headed --browser chrome

>npx cypress run --headed --browser chrome
```

To run cypress tests in different browser run below commands in terminal in headed mode:

```
>npx cypress run --headed
OR
>npx cypress run --headed --browser electron
>npx cypress run --headed --browser chrome
>npx cypress run --headed --browser firefox
>npx cypress run --headed --browser edge
OR
>node_modules/.bin/cypress run --headed
>node_modules/.bin/cypress run --headed --browser electron
>node_modules/.bin/cypress run --headed --browser chrome
>node_modules/.bin/cypress run --headed --browser firefox
```

```
>node_modules/.bin/cypress run --headed --browser edge
```

To resolve cypress automation/ test runner failing to open

npm uninstall cypress
npm install cypress --save-dev

Note:

```
Cypress only supports CSS selectors(JQuery is almost
considered as css selector only0
```

Css Selector

HTML

```
<input type="search" placeholder="Search for Vegetables and
Fruits" class="search-keyword" style="" xpath="1">
```

Syntax:

```
tagname[attribute=value]
.classname
#id
```

Ex:

input[type="search"]
input[placeholder="Search for Vegetables and Fruits"]
input.search-keyword

Syntax to filter the invisible locator

```
Cy.get('locator:visible')
cy.get('.product:visible').should('have.length', 4)
```

each

Iterate through an array like structure (arrays or objects with a length property).
It is unsafe to chain further commands that rely on the subject after .each().

Syntax

```
.each(callbackFn)
```

Arguments

callbackFn *(Function)*

Pass a function that is invoked with the following arguments:

- value
- index
- collection
-

Examples

DOM Elements

Iterate over an array of DOM elements

```
cy.get('ul>li').each(($el, index, $list) => {
```

```
  // $el is a wrapped jQuery element
  if ($el.someMethod() === 'something') {
    // wrap this element so we can
    // use cypress commands on it
    cy.wrap($el).click()
  } else {
    // do something else
  }
})
```

The original array is always yielded

No matter what is returned in the callback function, .each() will always yield the original array.

```
cy.get('li')
  .should('have.length', 3)
  .each(($li, index, $lis) => {
    return 'something else'
  })
  .then(($lis) => {
    expect($lis).to.have.length(3) // true
  })
```

Cypress Asynchronous nature and its promise handling

- Basically JavaScript is asynchronous.
- Any tool/ programming languages built on top of node.js (which is javascript) is asynchronous.
- Cypress is built on node.js which uses javascript language and is asynchronous in nature.
- Selenium Java is synchronous where we observe sequence of execution
- Protractor is asynchronous
- WebdriverIO is asynchronous

In general,
If it is synchronous, every step will execute in sequence manner.
If it is asynchronous, all the steps will hit the server at a time. (i.e., it will not wait for the previous step/ statement execution to complete instead it will execute subsequent steps at a time)

Ref: https://docs.cypress.io/guides/core-concepts/introduction-to-cypress#Commands-Are-Asynchronous

Commands Are Asynchronous

It is very important to understand that Cypress commands don't do anything at the moment they are invoked, but rather enqueue themselves to be run later. This is what we mean when we say Cypress commands are asynchronous.

Promise

- In Asynchronous, every step returns '**Promise**'. Every step returns a 'Promise'.
  Promise is nothing but the state or behaviour of step. Like what state our step is in
  -Executed
  -Pending

Promise comes 3 different states.

1. Rejected
2. Resolved
3. Pending

- Every Asynchronous step returns a Promise and its state will be either rejected, resolved or pending.
- Cypress is asynchronous in nature and there is no guarantee in sequence of execution but cypress take care of it

What is a promise in JavaScript?

The Promise in JavaScript may look quite complicated to understand at first sight, but in reality, it is quite simple and is not rocket science. In JavaScript, a promise is just like a promise that you make in real life to show that you are committed to doing something.
For example, I promise to get good marks in mathematics, and then this Promise has two outcomes, either it will be fulfilled (or resolved) or not fulfilled (or be rejected). So if I get good marks, the Promise is resolved, but if I fail to get good marks, it will not be resolved because I could not keep my Promise. However, in JavaScript, a promise has three outcomes; Promise gets resolved, gets rejected, or pending state, which means the Promise is not completed yet but may get completed after some time, so it is not rejected till now and is in the pending state.
Ref: https://www.javatpoint.com/what-is-a-promise-in-javascript

- Cypress will take care of the steps/ statements to execute in sequential order.
- Every asynchronous step returns 'Promise' and we have to wait until promise is resolved.
- When Promise is 'resolved', then we can be assured that step is executed and will move to next step.
- If cypress haven't take care of execution order sequentially, how we can know that 'Promise' is resolved?

Using **then()** method we can concatenate to the step and then() will wait until 'Promise' is resolved which means this step is executed and will move to next step.

Ref: https://www.browserstack.com/guide/cypress-async-tests
Example:

Example:

The example of asynchronous behavior in Cypress is provided below to help you comprehend the idea.

```
describe('Cypress', function () {
it('Example 1', function (){
// launching Url URL
cy.visit("https://example.cypress.io/")
// identifying element
cy.get('h1').should('have.text', 'Kitchen Sink')
cy.get('h1').then(function(e){
const t = e.text()
// get in Console
console.log(t)
})
// Console message
console.log("Cypress Tutorial")
})
})
```

Promises to handle Cypress Asynchronous Behavior

Before beginning to execute any commands, Cypress puts them all in a queue. Programming language promises are quite similar to general language promises made to humans. A promise is a condition that represents a person's behavior or activity.

Depending on the circumstance and nature, a person can either keep or break the Promise. When the Promise takes place, it is in an irrational state that might either resolve to be fulfilled or become refused.

On the same note, Promise is a state of the command in the case of Cypress async tests, and it will have the following states:

- **Resolved:** If the test phase is successful.
- **Pending:** If the test phase run result is being awaited.
- **Rejected:** If the test phase is unsuccessful.

A Cypress command executes only when the previous step has been executed successfully, or a resolved promise response has been obtained. Then, Promise is added to Cypress using the method.

Example of a Promise in Cypress:

```
describe('Cypress Test', function () {
it('Promise', function (){
return cy.visit('https://example.cypress.io/')
.then(() => {
return cy.get('h1');
})
})
})
```

then() method:

.then()

A command may occasionally yield a subject rather than return it. In that situation, we use .then() to communicate with the subject directly.

Promises and .then() act identically. However, unlike a Promise, .then() is a Cypress command. As a result, we cannot use async/await in the test script.

The callback function's output becomes the new subject and feeds the command that follows (except for undefined).

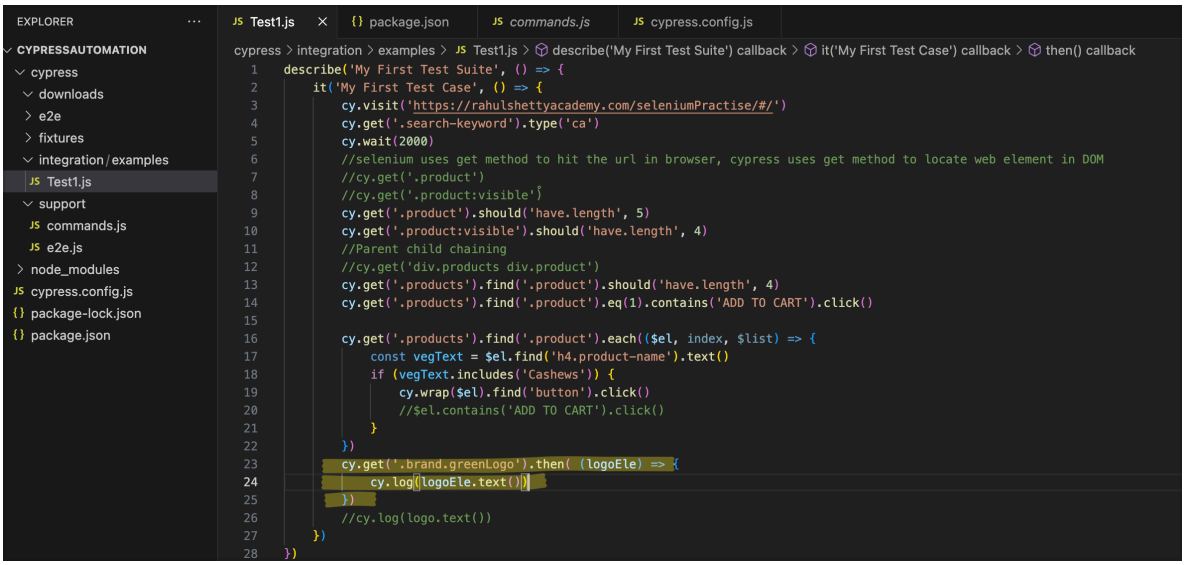
```
cy.get("button").then(($btn) => {
const cls = $btn.attr("class")
// ...
})
```

jQuery text() Method

The text() method sets or returns the text content of the selected elements. When this method is used to return content, it returns the text content of all matched elements (HTML markup will be removed). When this method is used

to set content, it overwrites the content of ALL matched elements.

- Cypress commands yields jQuery objects, so you can call methods on them
- **text()** -> is a jquery method and not the cypress method.
- Non cypress commands cannot resolve promise by themselves. Hence we need to resolve the Promise using then()



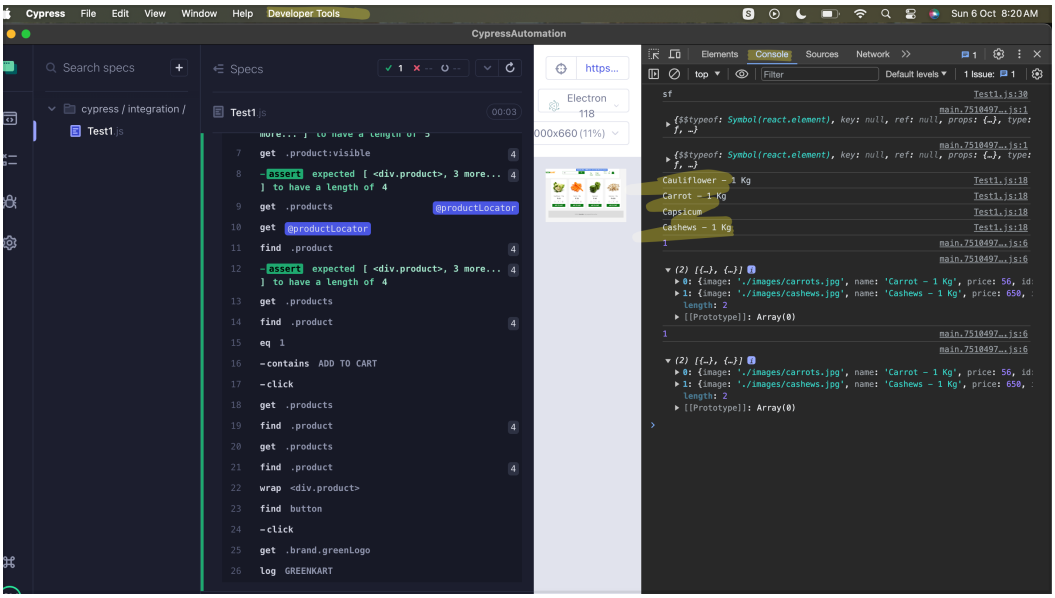
Alias:

- Aliasing can be done using Cypress ‘as’ command:
- Aliasing is to reuse locators

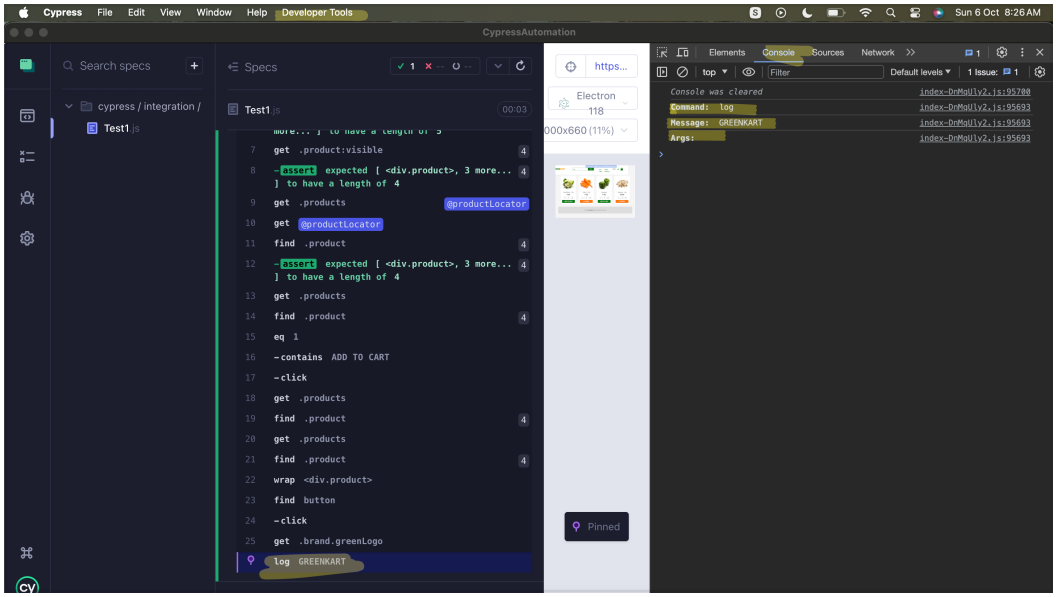
```
cy.get('.products').find('.product').should('have.length', 4)
||
  cy.get('.products').as('productLocator')

cy.get('@productLocator').find('.product').should('have.length', 4)
```

Console.log():

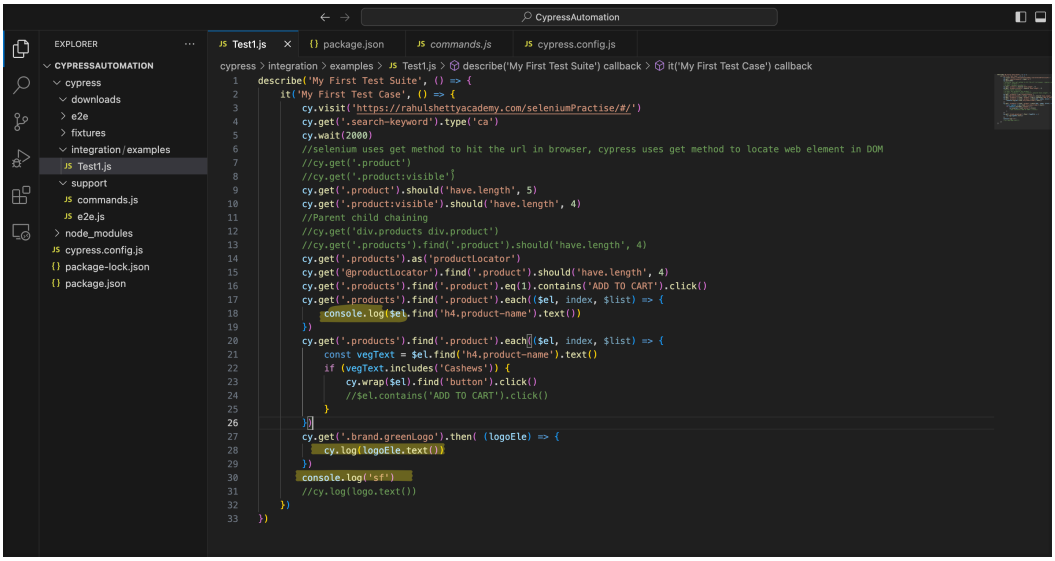
- Output gets printed in developer console of browser (electron browser)
- Console.log(‘sf’) -> is a javascript printing option and not cypress command
- Console.log() -> is asynchronous and hence printed immediately in console as there is no sequential execution of steps



Cy.log()

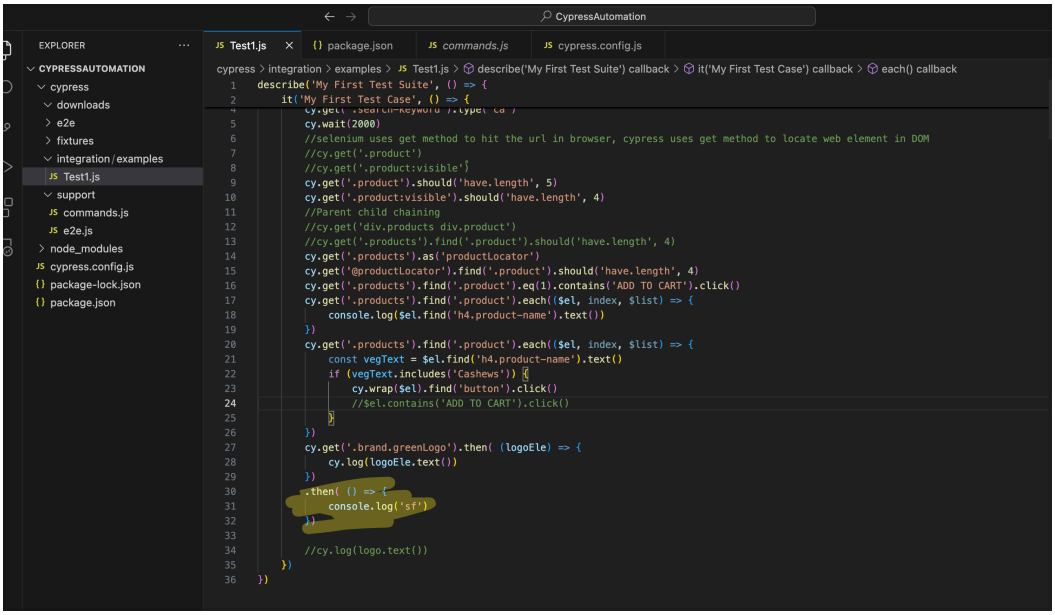
- Cy.log() is a cypress command
- Cy.log() is synchronous and hence printing in last as it this step executes in sequential manner
- 

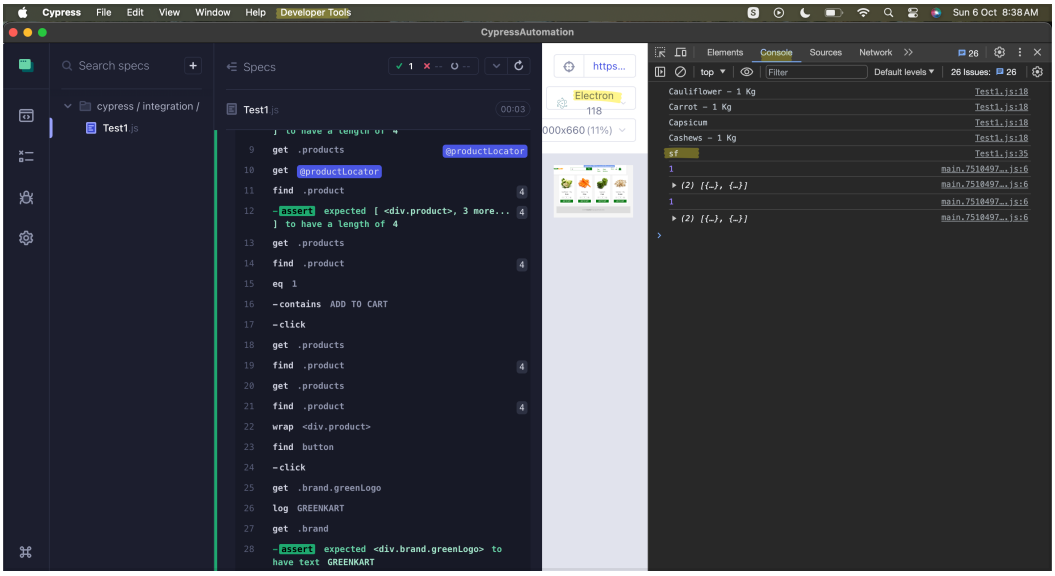
Console.log() & Cy.log()



To resolve the Promise, we manually need to add .then() in our code

```
.then( () => {
            console.log('sf')
        })
```





Code1.js:

```
describe('My First Test Suite', () => {
   it('My First Test Case', () => {
      cy.visit('https://rahulshettyacademy.com/seleniumPractise/#/')
      cy.get('.search-keyword').type('ca')
      cy.wait(2000)
      //selenium uses get method to hit the url in browser, cypress uses get
method to locate web element in DOM
      //cy.get('.product')
      //cy.get('.product:visible')
      cy.get('.product').should('have.length', 5)
      cy.get('.product:visible').should('have.length', 4)
      //Parent child chaining
      //cy.get('div.products div.product')
      //cy.get('.products').find('.product').should('have.length', 4)
      cy.get('.products').as('productLocator')
      cy.get('@productLocator').find('.product').should('have.length', 4)
      cy.get('.products').find('.product').eq(1).contains('ADD TO CART').click()
      cy.get('.products').find('.product').each(($el, index, $list) => {
         console.log($el.find('h4.product-name').text())
      })
      cy.get('.products').find('.product').each(($el, index, $list) => {
         const vegText = $el.find('h4.product-name').text()
         if (vegText.includes('Cashews')) {
            cy.wrap($el).find('button').click()
            //$el.contains('ADD TO CART').click()
         }
      })
      cy.get('.brand.greenLogo').then((logoEle) => {
         cy.log(logoEle.text())
      })
      //this is to assert the logo text
      cy.get('.brand').should('have.text', 'GREENKART')
```

```
        //this is print logs in console resolving the promise
        .then(() => {
            console.log('sf')
        })

    //cy.log(logo.text())
  })
})
```

Code2.js:

```
describe('My Second Test Suite', () => {
  it('My Second Test Case', () => {
    cy.visit('https://rahulshettyacademy.com/seleniumPractise/#/')
    cy.get('.search-keyword').type('ca')
    cy.wait(2000)
    //selenium uses get method to hit the url in browser, cypress uses get method to locate web element in DOM
    cy.get('.products').as('productLocator')
    cy.get('@productLocator').find('.product').eq(1).contains('ADD TO CART').click()
    cy.get('@productLocator').find('.product').each(($el, index, $list) => {
      const vegText = $el.find('h4.product-name').text()
      if (vegText.includes('Cashews')) {
        cy.wrap($el).find('button').click()
      }
    })
    cy.get('.cart-icon > img').click();
    cy.contains('PROCEED TO CHECKOUT').click()
    cy.contains('Place Order').click()
  })
})
```

`Shift + Command + K   => delete line in VS code -> MAC`

Handling alerts, pop ups in Cypress

- Cypress auto accepts alerts and pop ups
  - To grab alert text and validate we need to use cypress events
- Only Cypress has ability to interact and modify the DOM as there is no browser specific drivers needed
- Cypress has ability of browser events

Cypress events:

https://docs.cypress.io/api/cypress-api/catalog-of-events

Alert handling in cypress:

Event	Details
Name:	window:alert
Yields:	the alert text **(String)**
Description:	Fires when your app calls the global window.alert() method. Cypress will auto accept alerts. You cannot change this behavior.

- Fires when your app calls the global window.alert() method. Cypress will auto accept alerts. You cannot change this behaviour.
- Cypress have capability of browser events. window.alert is the event which gets fired on alert open. So we are firing this event through cypress to get access to that alert

```
 cy.on('window:alert', (str) => {
            //Mocha
            expect(str).to.equal('Hello , share this
practice page and share your knowledge')
 })
```

Confirm pop up handling in cypress

Event	Details
Name:	window:confirm
Yields:	the confirmation text **(String)**
Description:	Fires when your app calls the global window.confirm() method. Cypress will auto accept confirmations. Return false from this event and the confirmation will be canceled.

```
 cy.on('window:confirm', (str) => {
            //Mocha
            expect(str).to.equal('Hello , Are you sure you
want to confirm?')
})
```

Handling Child window and New tab:

We need to remove target attribute of the element in DOM and then only we can handle child or new tab

- jQuery methods by default available to Cypress
- Here invoke() is jQuery method to remove the attribute of a element present in DOM

```
cy.get('#opentab').invoke('removeAttr', 'target').click()
```

Ref: Cypress web automation — Handling Child Tabs - 3 approaches explained

Handling cross domain issue in cypress:

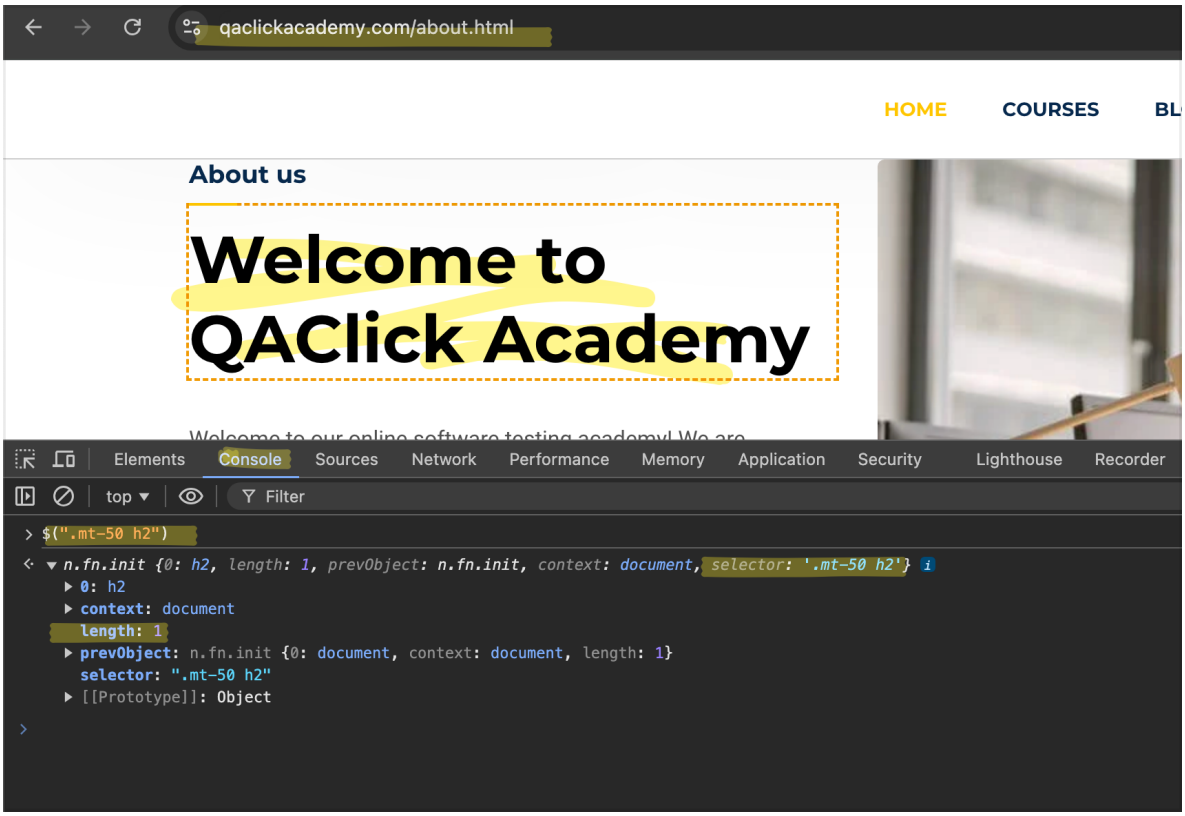
cy.origin() requires the first argument to be a different domain than top. You passed https://rahulshettyacademy.com to the origin command, while top is at https://rahulshettyacademy.com

Error if we don't handle cross domain issue:

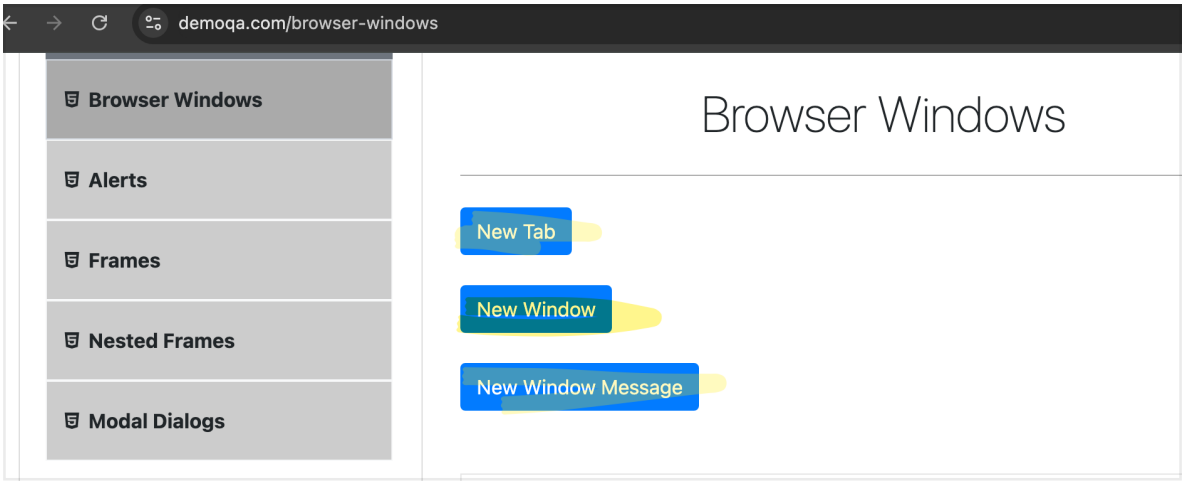
```
Timed out retrying after 4000ms: The command was expected to run against origin https://rahulshettyacademy.com but the application is at origin https://www.qaclickacademy.com.
```

Resolution:

```
cy.origin('https://www.qaclickacademy.com/', () => {
            cy.get("#navbarSupportedContent a[href*='about']").click()
            cy.get('.mt-50 h2').should('contain','QAClick Academy')
})
```



Handling child window, new tab, child window message without target, href attributes



Ref: https://stackoverflow.com/questions/69942630/cypress-how-to-handle-new-tab-without-target-and-href-attribute

```
describe('Example shows how to work with browser windows.',
() => {
    it('Example shows how to work witn button that opens new
tab without "target: _blank" and "href" attributes.', () =>
{
        cy.visit('https://demoqa.com/browser-windows', {
          onBeforeLoad(win) {
            cy.stub(win, 'open')
          }
        });

        cy.get('#tabButton').click();
        cy.window().then(() => {
        cy.visit('https://demoqa.com/sample',
{ failOnStatusCode: false })
        })
        cy.get('#sampleHeading').should('have.text','This is
a sample page')
        cy.go(-1)
        cy.get('#windowButton').click();
        cy.window().then(() => {
        cy.visit('https://demoqa.com/sample',
{ failOnStatusCode: false })
        })
        cy.go(-1)

        cy.get('#msgWindowButtonWrapper').click();
        cy.window().then(() => {
        cy.visit('https://demoqa.com/sample',
{ failOnStatusCode: false })
        })
    });
  });
```

Handling web table in Cypress:

- You can traverse to sibling with next() cypress method and it works only on get()

```
describe('Handling web table Suite', () => {
    it('Handling web table window', () => {
        cy.visit('https://rahulshettyacademy.com/
AutomationPractice/')

        cy.get('table[name="courses"]>tbody>tr>td:nth-
child(2)').each(($el, index, $list) => {
            const courseText = $el.text()
            if (courseText.includes('Python')) {

cy.get('table[name="courses"]>tbody>tr>td:nth-
child(2)').eq(index).next().then( (priceEle) => {
                    const priceValue = priceEle.text()
                    expect(priceValue).to.eq('25')
                })
            }
        })
    })
})
```

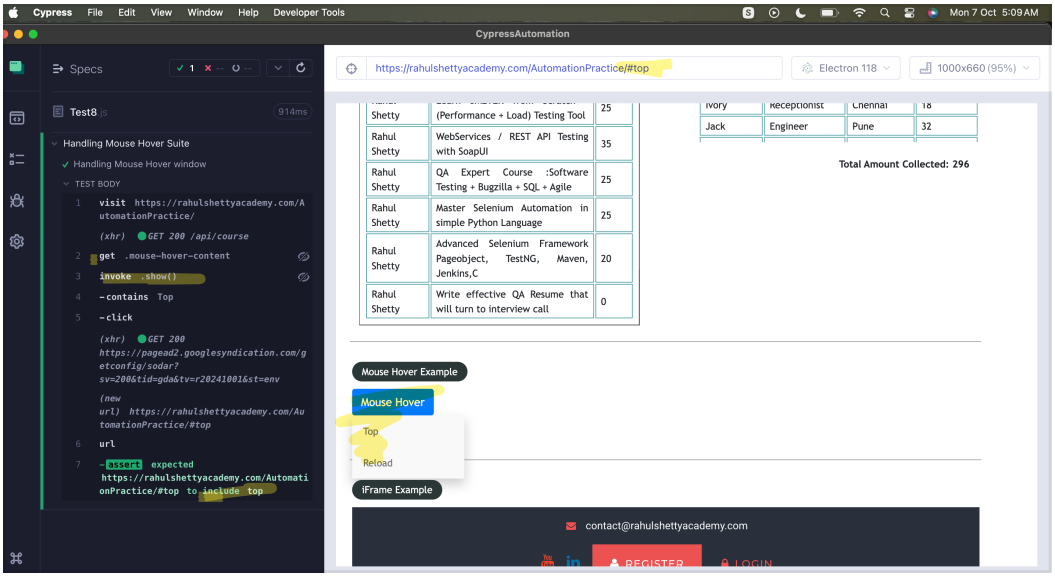
Handling mouse hover in Cypress

- Cypress cannot handle mouse hover
- Instead we have jQuery support to resolve the problem
- **jQuery Effect show() Method**

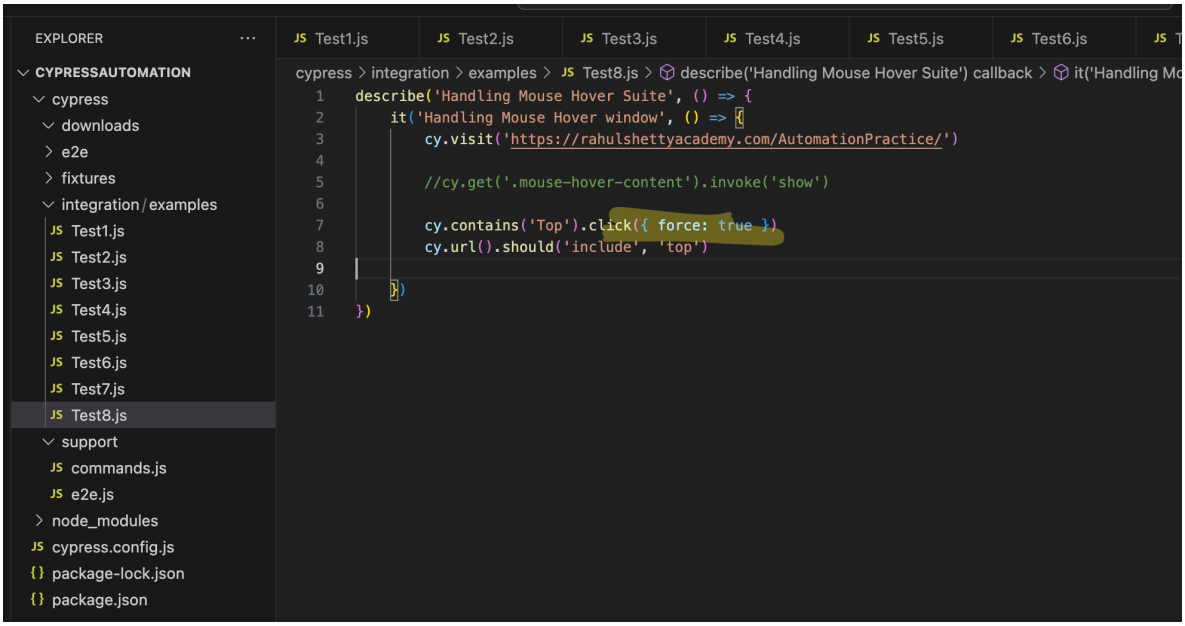
The show() method shows the hidden, selected elements. Note: show() works on elements hidden with jQuery methods and display:none in CSS (but not visibility:hidden). Tip: To hide elements, look at the hide() method.

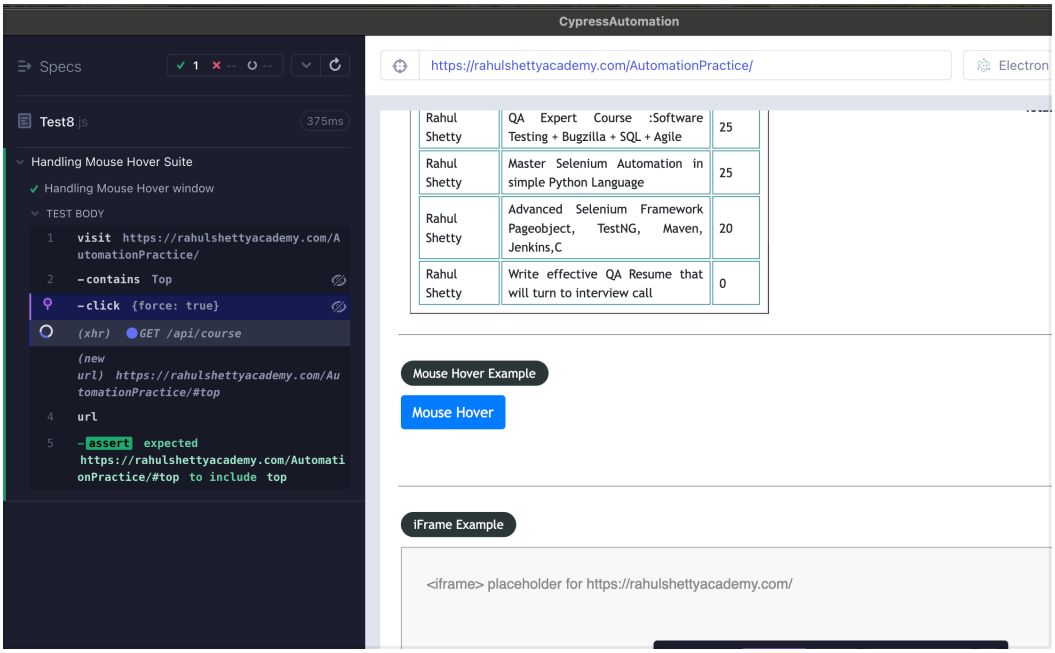
- Any JQuery method can be called using **invoke()** method

```
cy.get('.mouse-hover-content').invoke('show')
```



- Using .click({ force: true}) we can click on the hidden element present in mouse hover without the show()/ invoke() method of jquery (only if our operation is click)





Grabbing attribute value of element and navigate to new tab:

- JQuery prop() method provides a way to explicitly retrieve property values
- Below code works fine when we do not switch between domains like https://rahulshettyacademy.com & https://qaclickacademy.com

```
describe('Grabbing attribute value Suite', () => {
    it('Grabbing attribute value window', () => {
        cy.visit('https://rahulshettyacademy.com/
AutomationPractice/')

        cy.get('#opentab').then( (el) => {
            const url = el.prop('href')
            cy.visit(url)
        })

    })
})
```

- Below code works fine when we switch between domains like https://rahulshettyacademy.com & https://qaclickacademy.com

```
describe('Grabbing attribute value Suite', () => {
    it('Grabbing attribute value window', () => {
        cy.visit('https://rahulshettyacademy.com/
AutomationPractice/')

        cy.get('#opentab').then( (el) => {
            const url = el.prop('href')
            cy.visit(url)
```

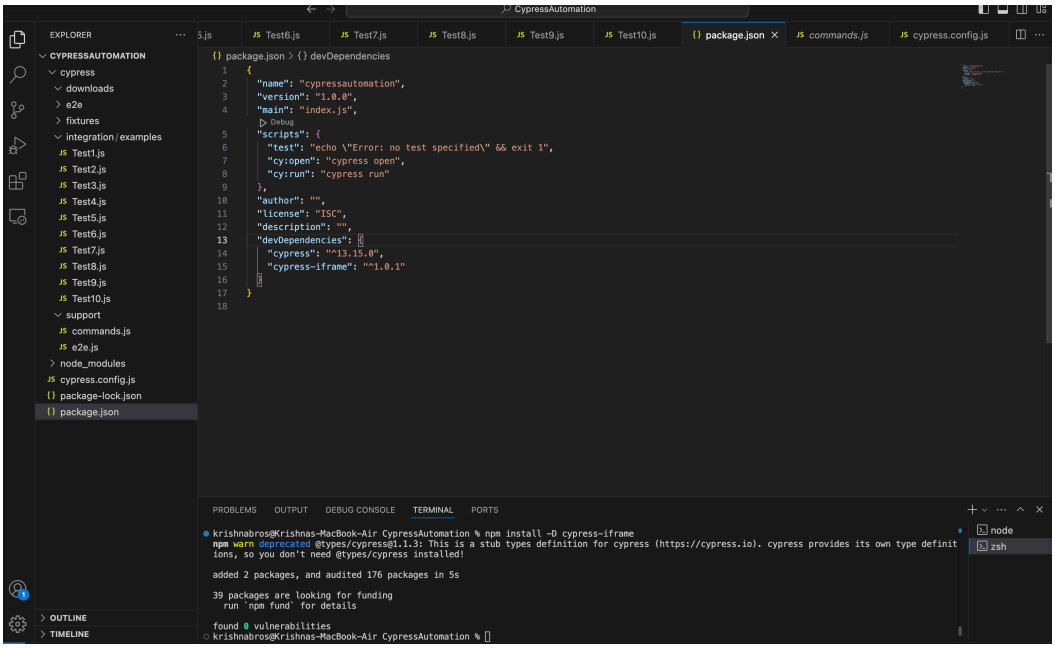
```
            cy.origin(url, () => {
                cy.get('div.sub-menu-bar
a[href*="about"]').click()
                cy.get(".mt-50
h2").should('contain','QAClick Academy')
            })
        })
    })
})
```

iFrames: Frames is nothing HTML document embedded in another HTML document

Handling Frames with Cypress

- Run below command to install cypress iframe to handle iframe

```
>npm install -D cypress-iframe
```



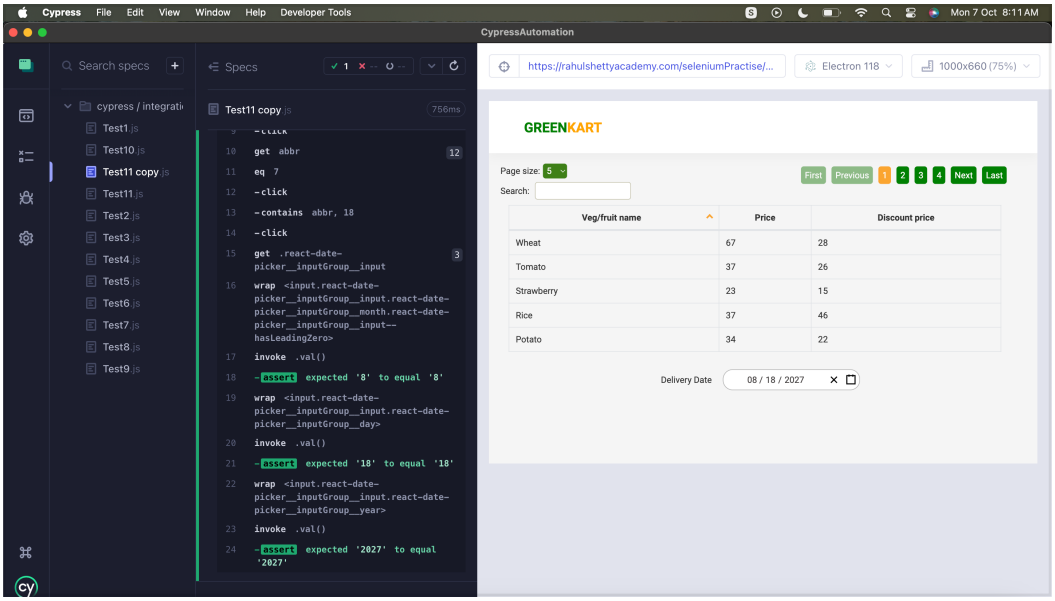

```
import 'cypress-iframe'
describe('Frames Test Suite', () => {
    it('Demo example', () => {
        cy.visit('https://rahulshettyacademy.com/
AutomationPractice/')

        cy.frameLoaded('#courses-iframe')

cy.iframe().find('a[href*="mentorship"]').eq(0).click({force
: true})
        cy.iframe().find('h1[class*="pricing-
title"').should('have.length', 2)
```

```
    })
})
```

Handling calendar in cypress

- Approach1

```
describe('Calendar Test Simple', () => {
    it('Verify date selection', () => {
        const monthNumber = '8'
        const day = '18'
        const year = '2027'

        const expectedList = [monthNumber, day, year]

        cy.visit('https://rahulshettyacademy.com/
seleniumPractise/#/offers')

        cy.get('.react-date-picker__inputGroup').click()
        cy.get('.react-calendar__navigation__label').click()
        cy.get('.react-calendar__navigation__label').click()
        cy.contains('button', year).click()
        cy.get('abbr').eq(Number(monthNumber) - 1).click()
        cy.contains('abbr', day).click()

        cy.get('.react-date-
picker__inputGroup__input').each(($el, index) => {
            cy.wrap($el).invoke('val').should('eq',
expectedList[index])
        })
    })
})
```



- Approach2:

```
describe('Calendar Test Suite', () => {
    it('Verify date selection', () => {
        const year = '2027'
        const month = 'August'
        const day = '18'

        cy.visit('https://rahulshettyacademy.com/
seleniumPractise/#/offers')

        cy.get('.react-date-picker__calendar-
button').click()
        cy.get('.react-
calendar__navigation__label').as('calenderEle')
        cy.get('@calenderEle').click()
        cy.get('@calenderEle').click()

        function selectUserInputInDate(userInput,
parameter2, parameter3) {
            cy.get('.react-calendar__tile').each(($el,
index, $list) => {
                const text = $el.text()
                if (text === userInput) {
                    cy.wrap($el).click()
                }
            })
        }

        selectUserInputInDate(year)
        selectUserInputInDate(month)
        selectUserInputInDate(day)
```

```
        cy.get('[name="date"]').then((el) => {
            const dateValue = el.prop('value')
            expect(dateValue).to.eq('2027-08-18')
        })
    })
})
```