Cypress interview questions for Freshers

1. How can you define the Cypress testing tool?

Cypress is a Javascript-based front-end testing tool. It has been written in node js. The tool is helpful in executing tests in a browser. This way, it can help make the entire testing process more reliable and much easier.

2. How will you install Cypress?

To install Cypress, we will first have to install Node. once done, we can then install Cypress with the command - npm install cypress.

3. Can you define the Cypress architecture?

Behind this tool is a Node server process. The Node process and Cypress consistently synchronize, communicate and execute tasks on behalf of each other. Not just that, Cypress also operates at a network layer by altering and reading web traffic simultaneously. This helps Cypress in modifying everything coming from the browser. In addition to this, the tool also becomes capable of changing code that could interfere with its capability to automate the entire browser.

4. What are the primary features of Cypress?

Some of the major features of Cypress include:

- This tool can run tests and execute commands on the browser; thus, the tests are much fast and less flaky.
- It can take snapshots of tests after the execution of every step. This way, you won't need any additional plugins.
- Cypress also offers network traffic control, inbuilt assertions, and automatic wait.

5. What are the advantages of Cypress?

Some of the significant advantages of Cypress include:

- Cypress works with Internal Ajax Calls and Single Page Applications.
- It is capable of executing commands easily and faster in comparison to tools based on Selenium.
- This tool can easily take screenshots of tests; thus, we don't need to have any configuration with it.
- It can run tests and execute commands on the browser.

6. What are the disadvantages of Cypress?

Some of the major drawbacks of Cypress include:

- It only supports JavaScript.

- It is only available for web testing.
- It doesn't support multiple tabs.
  It doesn't support Iframes.

7. What do you mean by Cypress CLI?

Cypress CLI is helpful in offering the capability for running the Cypress tests in Command-Line. It is one such feature that is used to run the Cypress tests in the Pipelines. Moreover, it also offers Flags Control and Options of the Cypress Test Behaviour.

8. Name some browsers that Cypress supports.

Cypress supports:

- Electron
- Firefox
- Edge
- Chromium
- Chrome

9. How can you differentiate Cypress from Selenium?

- Cypress supports just Typescript and JavaScript languages. On the other hand, Selenium supports all the major languages, such as C#, Python, Java, Ruby, JavaScript, and more.
- While Selenium supports all major browsers, Cypress supports just a few of them.
- In Selenium, the configuration is difficult. In Cypress, you can find ready-to-use frameworks.

10. What are the testing frameworks that come with Cypress?

Cypress is available with Chai and Mocha assertion libraries. However, we cannot use the TestNG or Junit in Cypress.

11. How can you interact with DOM elements in Cypress?

We can use CSS Selector to interact with DOM elements in Cypress.

12. Can you use BDD with Cypress?
Cypress doesn't provide any official inbuilt support for BDD. However, the NPM Cypress-Cucumber-Preprocessor plugin lets you write tests in BDD Cucumber Syntax. And then, it will automatically translate the same into Cypress.

13. What do you mean by hooks in Cypress?
Cypress hooks are generally used to set or define preconditions that we wish to execute either before every test or before a set of tests. Some of the available hooks are afterEach(), after ( ), beforeEach ( ), and before ( ).

14. How will you manage reusability in the Cypress framework?
Cypress comprises, by default, the commands.js and index.js in the Support folder. Index.js runs before each test file. We can put the reusable behaviour, such as global override and custom commands in this folder.

15. Define the components of Cypress.
Jotted down below are the major components of Cypress:

- **Test Runner:** It tests in an interactive runner, which further helps by letting you see the command and execute the same while viewing the application that is under the test.
- **App Preview:** It helps in seeing the tests while executing the commands.
- **Test Status:** It assists in displaying a summary of what tests are in Progress and have Failed or Passed.
- **Command Log:** It showcases the command logs while executing the tests.
- **URL Preview:** It displays the URL of the test and assists in tracking the entire URL Route.
- **Viewport Sizing:** It helps in setting the app viewport size for testing varying layouts.

16. What do you mean by automation in Cypress?
Cypress automation is a process that runs a Test Code with the app. This test can't be executed in a single loop. However, it takes place in a Single Browser. Node.js Server is generally used in the tasks that happen outside.

17. Define Cypress run.
Cypress Run assists in executing the Cypress tests in Headless Browsers. It also assists in opening a New Browser Tab. along with that, it's also helpful in loading the tool at the URL Cypress, which was earlier installed from the same. Cypress assists in discovering cypress.json. Moreover, it continues to run tests on the webpack monitor.

18. What selectors does Cypress support?

By default, this tool only supports CSS selectors. But, we can also use third-party plugins to use the Xpath selectors.

19. How will you open the Cypress window and execute tests?

Once we have installed Cypress, we can open it using the npm cypress open command. And then, by clicking on the test file name, we can execute the tests.

20. How to see the default configuration in this tool?

The default configuration can be seen Under on Cypress window Test Runner. There, you will have to click Settings and Configurations.

Cypress Interview Questions for Experienced

1. How will you create a test suite in Cypress?

We can develop a describe ( ) block as it acts as a suite. Inside this block, every test can be created by using a single it ( ) block.

```
describe('Test Suite', () => {
    it('Tc01', () => {
        //code
    })
})
describe('Test Suite', () => {
    it('Tc01', () => {
        //code
    })
})
```

2. Is it possible to use XPath in Cypress?

By default, Cypress doesn't support XPath. But, with the help of the Cypress-Xpath plugin, we can use XPath to interact with all the DOM elements. We can install the plugin through npm.

```
npm install cypress-xpath
```

3. How can you verify the title of a page in Cypress?

A title of a page in Cypress can be easily verified through the should assertion.

```
cy.title().should('eq','My Site Title')
```

4. How can you read values from the configuration file?

We can read the values of cypress.config file through **Cypress.config();**

For instance, if we have managed to define the env in the config.json file, such as:

```
{“env”: stage}
```

The same can be accessed in the script, such as:

```
let timeout = Cypress.config(‘env’)
```

5. How do you maintain the test data in Cypress?

Generally, the fixture folder has the important data in the Cypress project. It helps in getting the data input from all the external files. All of the test data can get used for multiple tests. Also, every bit of fixture data should be declared within the before hook block.

```
cy.fixture(path of test data)
```

6. How will you verify whether a button is visible or not?

It can be verified using this command:

```
cy.get('button#form-submit').should('be.visible')
```

7. Define the types of assertions in Cypress.

There are negative and positive assertions in Cypress.

- **Positive Assertions**

  ```
  cy.get('li.todo').should('have.length', 3)
  ```

- **Negative Assertions**

  ```
  cy.get('li.todo').should('not.have.length', 2)
  ```

8. How will you perform API testing in Cypress?

The API testing in Cypress can be performed using this command:

```
describe('API Testing in cypress', function () {

   it('Hit Get Request validate its response status code and body', () => {
      cy.request({
         method: 'GET',
         url: 'https://randomuser.me/api/',
         qs: 'results=1'
      }).then((response) => {
         expect(response.status).to.eq(200)
         expect(response.body).to.have.property('info')
       })
   })
})
```

9. How will you run single specfile in the command line?

The Single Specifile can run through the spec option and using the command:

```
npm cypress run - - spec="my spec.ts""
```

10. How will you access shadow DOM in Cypress?

Shadow DOM generally helps in letting hidden DOM Trees that has to be attached to the elements in a regular DOM Tree. Shadow DOM can be comprehended through this command:

```
cy.get('#locator').shadow().find('.nb-btn').click()
```

11. How will you verify the title of a page in Cypress?

It can be verified through this command:

```
cy.title().should('eq','My Site Title')
```

12. How will you press keyboard keys in Cypress?

The keyboard keys can be pressed through this command:

```
cy.get('#locator').type('{shift}{alt}'));
```

13. How can you create custom commands in Cypress?
The custom commands can be used through the following command:

```
Cypress.Commands.add("login", (username, password) => {
  cy.get("#username").type(username);
  cy.get("#password").type(password);
  cy.get("#login").click();
});
```

14. How will you change the baseUrl dynamically?
You can change the baseUrl dynamically through this CLI command:

```
npx cypress run --config baseUrl="
```

1**5. How will you create suites in Cypress?**A describe() block can be created as it acts as a suite. Inside that, every test can be created as a single it () block.

16. Can you define some Cypress commands?
Some of the Cypress commands are:

- **To navigate to a certain site**

  ```
  cy visit(): cy.visit()
  ```

- **To showcase Cypress console logs during execution**

  ```
  cy.log (): cy.log
  ```

- **To get DOM elements in Cypress**

  ```
  cy.get(): cy.get
  ```

- **Get the Current URL of a page that is active currently**

```
cy.url(): cy.url()
```

17. Can you list some commands that can be used to interact with DOM elements?

Here are some commands that can be used to interact with DOM elements:

- **To click on an element**

```
.click()
```

- **To double click on an element**

```
.dblclick()
```

- **To right click on an element**

```
.rightclick():
```

- **To type on text boxes or element**

```
.type():
```

- **To clear the text boxes or fields**

```
.clear():
```

- **To check radio(s) or checkbox(es)**

```
.check():
```

18. Can you define an environment variable in Cypress?

The environment variables are referred to such variables whose value is set at a level of the operating system. It is set outside the content of a framework or a program. To use an environment variable, we have to define it in cypress.json.

```
{
  "env": {
            key: value
  }
}
```

Once we have defined, we can access them in the script just the way mentioned below:

```
cy.visit(Cypress.env('key'));
```

This environment variable value can be changed dynamically through command line arguments:

```
cypress run --env "Key"="Value"
```

Let’s consider an example. Here, you can find an environment variable created in cypress.json.

```
{
  “env”: {
      token: “ueidkskslsdfjsdlf”
    }
}
```

This environment variable’s value can be accessed via:

```
let tokenvalue = Cypress.env(‘token’);
```

If you wish to change the value dynamically, it can be done using this command line:

```
cypress run --env "token"="ieoeeoeieoeie";
```

19. How will you define preserve cookies? Why are they important?

By default, Cypress clears cookies after each test. To stop clearing these cookies, we use preserve cookies in Cypress. For instance:

Here is a code:

```
describe('my test', () => {
it('test1', () => {
//Some code to test
});
it('test2', () => {
//Some code to test
});
it('test3', () => {
//Some code to test
});
})
```

In the code mentioned above, we've got three tests, such as test1, test2, and test3.

Let's suppose you've logged into your app in test1. After executing the test1, when it goes to test2, it again asks to login as login sessions and cookies were cleared after test1.

All of this happens by default in Cypress. This problem can be handled by adding a code in the test as mentioned below:

```
describe('my test', () => {
beforeEach(() => {
Cypress.Cookies.preserveOnce('session_id', 'remember_token')
});
it('test1', () => {
//Some code to test
});
it('test2', () => {
//Some code to test
});
```

```
it('test3', () => {
//Some code to test
});
})
```

20. How will you get the first as well as the last child of a chosen element?

The first and the last child of a chosen element can be acquired through .first() and .last() command. For example:

```
cy.get('input[name='rows'])
```

Let's suppose these selectors return several elements, for instance, say six of them. So:

The first child can be obtained through:

```
cy.get('input[name='rows']).first()
```

The last child can be obtained through:

```
cy.get('input[name='rows']).last()
```