1. What is Playwright?

Playwright is an open-source testing tool, which supports functional, API, and component testing. The playwright is managed by Microsoft.

2. What is the difference between Selenium and Playwright?

The playwright is ready to use a framework one can install and start using.	Selenium Provides API/Libraries you need to build the framework
Playwright shipped with in-built assertion libraries	Selenium doesn't provide any assertions, we need to integrate using JUnit or TestNG
The playwright uses the WebSocket connection to communicate with the browser	Selenium uses the Webdriver API/HTTP to communicate with the browser
The playwright is faster compared to Selenium	Selenium comparatively slower
The playwright doesn't support the safari stock browser rather it uses the open-source, Webkit browser to test safari	Selenium supports Safari
Playwright officially supports Java, Python, .NET C#, TypeScript, and JavaScript.	Selenium officially supports Java, Python, C#, Ruby, Perl, PHP, and JavaScript

3.What are the advantages of a Playwright?

Compared to any other framework Playwright has a lot of advantages, as it is a modern solution it's built on top of the limitation of another framework

- The playwright is easy to install and learn
- Playwright supports Java, Python, .NET C#, TypeScript, and JavaScript.
- It supports both API and end-to-end testing
- Playwright supports Chromium-based browsers, Firefox, Safari(Webkit)
- As Playwright doesn't use the webdriver API the execution is comparatively faster
- Playwright automatically waits before making any actions where a user doesn't have to provide implicit or explicit waits
- Playwright allows Network traffic control. Mocks etc.
- Edge case scenarios like File upload and download can be handled easily in playwright

4. Name some disadvantages of Playwright.

- Playwright doesn't support Mobile automation (They might introduce it in the future)
- Playwright doesn't support legacy IE Automation
- Playwright doesn’t support Safari stock browser
- Some of the build tools like Teamcity is not directly support
- Some of the features like Ordering, Priority, and Dependancy tests which are available in TestNG are not available in Playwright yet.

5. What are the different testing types the Playwright supports?

Playwright supports functional testing, API testing, and Component level testing.

6. What are the programming languages that the playwright supports

Playwright supports Java, Python, .NET C#, TypeScript, and JavaScript. However, the Typescript/Javascript version of Playwright is more stable and most used.

7. Briefly describe the commands that are used for Playwright installation and Execution of tests

As Playwright supports many programming languages each programming language follows its own installation process.

In this context we are using the Playwright and Javascript we need to use the following commands

Before installation, we need to ensure that NodeJS binaries are installed in our machine and then we can use

npm init playwright@latest

The above command will install the required packages and configurations. Once done we are ready to write the test cases.

npx playwright test

The command is used for executing playwright tests. By default, the playwright executes all the tests that are available for the project.

8. What is a Configuration File in Playwright explain?

As the name indicates the configuration file is the place where we configure the execution behavior of the playwright. The configuration file is typically named playwright.config.ts(/js).

Using the configuration file we can configure headless mode, fullscreen, screenshot options, baseURL, browser options, etc.

9. What is @playwright/test package in Playwright?

The Playwright can be used with different test runners such as Mocha, Jasmine, Jest, etc. Similar way playwright has its own test runner called the playwright test. The playwright test is the default test runner for the playwright.

10. What is Page class in Playwright?

The Page class in playwright is the main class, which provides the methods to interact with the browser. It also provides a set of event handlers that helps to execute a set of actions as soon as the event triggers.

11. How to navigate to specific URLs in Playwright explain with sample tests

```
const { test, expect } = require("@playwright/test");

test.describe("navigation", () => {

  test("navigation", async ({ page }) => {

    await page.goto("https://playwright.dev/");

  });

});
```

The **page.goto()** is used for navigating to a specific URL.

The **test.describe()** hook is used for grouping the set of tests

The **test()** contains actual tests with playwright commands.

12. What are the different types of reporters that the playwright supports?

The playwright supports different types of reports

- Dot reporter
- Line reporter
- HTML reporter
- JSON reporter
- JUnit reporter
- Custom reporter

In addition to the above playwright also supports allure reporters using third-party plugins.

13. What are the locators in the Playwright list of any five

Locators help to find the elements on the page uniquely at any point in time.

The page class provides the locator function.

- page.getByText() : Find the element that matches the given text
- page.getByRole(): Find the element that matches the role attribute
- page.getByLabel(): Find the element that matches the label
- page.getByTestId(): Find the element that matches the data-testid attribute
- page.getByTitle(): Find the element that matches the title attribute
- page.locator(<css> or <xpath>): Find the element by using the CSS or XPath

14. What are the different types of text selectors available in Playwright?

Text-based locators in Playwright are a unique feature, that helps to locate the element which is having specific text

locator.(<some_text) : Matches the element that has passed text

Ex: await page.locator('text=Home menu')

:text-is(): this can be used inside the CSS selector, which will perform the exact match before finding the elements

Ex: await page.locator('#nav-bar :text-is("Home")')

:has-text(): This is another pseudo-class, which matches the element which contains the passed text.

Example: await page.locator(':has-text("Playwright")')

15. How to use assertions in Playwright? List any 5

Playwright Test uses the jest expect library for assertions.

The Playwright supports soft and hard assertions both.

There are many assertions which expect library provides some of them are

expect(value1).toBe(value2): This helps to compare two values. Like equals of two strings etc.

expect(<Boolean_value1).toBeTruthy(): Assert two boolean values to be true.

expect(locator).toBeVisible(): Ensured specified locator is Visible on DOM

expect(locator).toContainText(expected_text): Ensures the specific DOM element contains the given text

expect(locator).toHaveClass(expected_class): Ensures the locator has specified css class

expect(locator).toHaveCount(count): Ensures the given locator count in dom should be equal to "count"

expect(page).toHaveTitle(title): Verifies the page title is the same as expected

16. What are soft assertions in Playwright?

By default when the assertions fail the test terminates, but if we use the soft assertions do not terminate the execution, but the test will be marked as failed at the end.

The Playwright provides a command

expect.soft() for soft assertions

Example:

expect.soft(page.locator('#title').toHaveText('Udemy')

17. How to negate the Matchers/Assertions in Playwright? Or How can I verify not conditions in Playwright?

The Negation matchers are most commonly used in Playwright.

The **.not** can be used in Playwright to Negate the assertions

For example, if we have to verify the a should not equal 10 then we can use

expect(a)not.toBe(10)

The not is generic keyword can be used with any assertions to negate the condition.

18. Does Playwright support XPath? If so how can I use them in the test?

Yes, Playwright supports both CSS and XPath selectors.

The **page.locator()** function can be used for XPath.

The page.locator() automatically detects the passed value is XPATH or CSS and returns the locator accordingly

Whenever Playwright sees the selector staring with // or .. then the playwright assumes it is the XPath

Example:

await page.locator("//button[@id='someid']").click();

19. What are command line options in the Playwright? Explain 5 helpful options

The configuration file contains the same set of run time configurations, the command line options also provide the run time configurations. When the same option is used in both places the command line options take priority.

20. Some of the Important command line options

- **Run all the tests**

npx playwright test

- **Run a single test file**

npx playwright test tests/todo-single.spec.ts

- **Run multiple tests**
- npx playwright test tests/todo-page/ tests/landing-page/

- **Run tests in headed mode**

npx playwright test --headed

- **Run tests on Specific browser**

npx playwright test --browser "chromium"

- **Retry failed test**
- npx playwright test --retries 2

21. What is headed and headless mode in Playwright

- The headed mode browser is just like any other browser. For example, if you open a chrome browser opens you can see it visually and perform the action on it.
- The headless browser execution happens in the background, the actions and browsers are cannot be seen. The headless browser execution is faster than the headed mode.

22. Does Playwright support HTML reporters? How to Generate HTML reports in Playwright?

Playwright supports default HTML reporter. You can generate the HTML reporters using the below command.

npx playwright test --reporter=html

23. What are timeouts in Playwright? What are the different types of Timeouts available in Playwright?

Timeout ensures that before marking test execution as a failure the amount of time the Playwright needs to wait.

In Playwright timeouts can be set at different levels

Test timeout: This is applicable for test and fixture execution time.

Expect timeout: Timeout for assertions

Action timeout: Timeout for each action

Navigation timeout: Timeout application for Navigation actions such as page.goto() etc.

Global timeout: Global Timeout for the whole test run

beforeAll/afterAll timeout: Timeout applicable for beforeAll/afterAll hooks

Fixture timeout: This timeout is set to individual fixtures.

24. How to navigate forward and backward in Playwright?

The playwright provides specific commands to navigate backward and forward.

page.goForward() command can be used to navigate forward from the browser history

page.goBack() command can be used to navigate backward from the browser history

25. How to perform actions using the Playwright?

The Page class provides different locator strategies. First, you need to find the element using locators and then perform actions.

For example, if you need to perform a click action on a button you can do it as shown below

await page.locator('#myButton').click();

26. Does playwright support the safari browser if so can we execute the test on safari?

Unlike selenium, Playwright doesn't support the Native safari browser. The Playwright supports the open-source version of Safari which is a Webkit browser.

You can execute the tests on the Safari Webkit browser using the configurations in the config file below:

```
const config = {

  use: {

    channel: 'chrome',

  },

};
```

By specifying in the command line parameter

```
npx playwright test --browser "webkit"
```

27. How to wait for a specific element in Playwright?

The playwright has an auto-waiting mechanism that reduces the usage of explicit waits in the test. However, in some scenarios, we might need to wait for specific elements in that case we can use the .waitFor()

Example:

```
const someElement = page.locator('#myElement);

await someElement.waitFor();
```

By default waitFor() waits for the element to be visible, however, the behavior can be changed to waitFor attached, detached, hidden

Example: await someElement.waitFor({ state: 'attached' })

28. What is browser context?

Browser context provides a way to operate multiple independent browser sessions. The browser class provides newContext() method which helps to create a new browser context.

Example:

```
const contxt = browser.newContext()
```

29. How to open multiple windows in Playwright?

The playwright provides a way to open multiple windows. In Playwright each window is called Pages.

You can create multiple pages like below

```
const pageOne = await context.newPage();

const pageTwo = await context.newPage();
```

30. How to handle iFrame in PLaywright?

The **frameLocator()** gets the frame on a particular page in Playwright. Using frameLocator() we can perform various actions on the iFrame

Example:

```
const button = await page.frameLocator('#my_ifr').locator('#button_my)
```

31. Explain some of the click and double click actions with its options.

click(): action is used for clicking any element on the page.

dblclick: The dblclick() is used to perform the double click on the element.

Both of the above click takes multiple parameters for example:

force: PLaywright waits for actionability it internally checks for if the element is ready to click. The force option helps bypass this option

locator.click({force:true});

position: Position can be used to perform the coordinates with respect to the element.

delay: can be used for time between mouseup and mousedown

32. How to perform a right-click on Playwright?

The playwright doesn't have a separate command for right click, in order to right click we need to use the click action with the button parameter

Example:

locator.click({button:right})

33. How to evaluate Javascript in Playwright?

The playwright provides page.evaluate() function, which can be used for evaluating any javascript function.

Example:

const href = await page.evaluate(() => document.location.href);

34. What are Playwright fixtures?

Test fixtures are used to establish an environment for each test. Some of the pre-defined fixtures are page, context, browser, browserName. The fixture is isolated for each test.

Consider you have multiple tests like below

```
test('basic test', async ({ page }) => {

  await page.goto('https://playwright.dev/');

});

test('basic click, async ({ page }) => {

  await page.locator('#logo').click();

});
```

Though you expect the above test to be executed one after the other, the test runs and fails. As the page fixture is isolated.

You can override the default fixtures like below.

For example, the above test can be rewritten to execute correctly

```
let todoPage;

test.beforeEach(async ({ page }) => {

    todoPage = new TodoPage(page);

  });

test('basic test', async () => {
  await todoPage.goto('https://playwright.dev/');

});

test('basic click, async () => {

  await todoPage.locator('#logo').click();

});
```

You can notice that the **todoPage** is shared between tests.

35. What is CodeGen in Playwright?

Playwright codeGen is similar to the selenium test recorder, the **CodeGen** is a tool that comes with playwright you can use it for recording the Playwright tests.

36. How to parameterize tests in Playwright?

Parameterize helps to run the same tests with multiple values, some times it is also called as data-driven testing. Playwright allows parameterization, you can use data from either csv, json or plain arrays. To implement parameterization you need to use for or foreach loop.

Example:

```
    const fruits = ['Banana', 'Orange','Apple'];

    for (const name of fruits) {

      test(`testing with ${name}`, async () => {

         //your code

      });
```

}

37. Write a code to upload the file

- The playwright provides a special command to upload a single file or multiple files. The command setInputFile() or setInputFiles() is used for uploading the file in Playwright.
- Example:
- **Upload single file:**

```
await page.getByLabel('Upload file ').setInputFiles('myfile.pdf');
```

- **Upload multiple files:**

```
await page.getByLabel('Upload files').setInputFiles(['file1.pdf', 'file2.pdf']);
```

- ***Note: Passing empty array to setInputFiles() makes unselect the files if you are already selected.***
- Example:

```
await page.getByLabel('Upload file').setInputFiles([]);
```

38. Write a code to download the file

- The upload and download files are edge case scenarios, Playwright has dedicated commands for both. The download files can be performed in the playwright using the waitForEvent() in the playwright.
- Example:

```
const [ download ] = await Promise.all([

  page.waitForEvent('download'),

  // Perform the action that initiates download
```

page.locator('button#delayed-download').click(),

]);

- Once the download is complete you can get the downloaded path using the command

const downloadedPath = await download.path();

39. How to perform drag and drop in Playwright?

- Drag and drop can be performed using multiple ways
- Using dragTo() command
- Manually specifying mouse actions

Using dragTo() function for drag and drop

You need to pass the target position locator to dragTo function

Example:

await page.locator('#item-to-be-dragged').dragTo(page.locator('#item-to-drop-at'));

The above dragTo() function internally does

- Hovers on the item to be dragged
- Clicks on the item to be dragged
- Move the mouse to target location
- Rleases the left mouse button

As mentioned earlier, you can use the manual method, by perfoming all the above actions instead of dragTo() function.

Manual way to drag and drop

await page.locator('#item-to-be-dragged').hover();

await page.mouse.down();

await page.locator('#item-to-drop-at').hover();

await page.mouse.up();

40. How to handle browser popups or dialogs?

- Dialog popups are native to the browser or operating systems. The dialogs need special mechanism to handle as you cannot inspect the locator for these pop-ups.
- There are different types of pop ups such as alert(), confirm(), prompt()
- **Handling Alert**
- //Click on Ok

page.on('dialog', dialog => dialog.accept());

- **Handling confirm**
- //Click on Ok

page.on('dialog', dialog => dialog.accept());

//Click on Cancel

page.on('dialog', dialog => dialog.dismiss ());

- **Handling Prompt**
- //Type the text, RSAcademy and Accept the pop up

page.on('dialog', dialog => dialog.accept("RSAcademy"));

41. What is testInfo Object?

testInfo object contains information about tests that are currently running such as duration, errors, project, status, etc. Using the testInfo object we can control the test execution.

42. What is testError Object?

The testError object in PLaywright contains information about errors thrown during the test execution such as error message, stack, and value.

43. What is global setup and tear down explain?

- The global setup is one-time setup that is needed for test execution. The global setup is executed before starting any tests. For example, if you want to set up some files, and URLs you can utilize this function.
- Similarly, the global teardown is a one-time teardown option provided by Playwright. The global teardown will be executed after all the tests are executed. This will be helpful to generate custom reports, sending emails, freeing up resources, etc.
- **Example Global Set up**

```
// global-setup.js

module.exports = async config => {

  const {storageState } = config.projects[0].use;

};

// playwright.config.js

const config = {
  globalSetup: require.resolve('./global-setup'),

  use: {

    storageState: 'state.json',

  },

};

module.exports = config;
```

- **Example Global Teardown**

```
// global-teardown.js

module.exports = async config => {

 //Some code

};

// playwright.config.js

const config = {

  globalTeardown: require.resolve('./global-teardown),

  use: {

    storageState: 'state.json',

  },

};

 module.exports = config;
```

44. How to capture Network logs in Playwright?

The playwright provides a way to monitor browser network logs. You can capture all the request and response network logs and their status. Using the listener

```
page.on('request', request =>

     console.log('>>', request.method(), request.url()));

  page.on('response', response =>

     console.log('<<', response.status(), response.url()));

  await page.goto('https://example.com');
```

45. How to capture screenshots in PLaywright?

- The Playwright allows taking the screenshot. the page.screenshot() function is provided by Playwright to the screenshot. You can place the screenshot() command anywhere in the code to save the screenshot.
- **Take the full page screenshot**

```
await page.screenshot({ path: 'screenshot.png', fullPage: true });
```

- **Take the Element level screenshot**

```
await page.locator('.header').screenshot({ path: 'screenshot.png' });
```

46. Does Playwright support API testing? If so how can we perform API testing?

Yes, Playwright supports API Testing. We can perform any HTTP API method calls such as GET, POST etc. using the playwright and validate the status and responses.

Example:

```
test("Get users", async ({ request, baseURL }) => {

  const apiResponse = await request.get(`${baseURL}public/v2/users/`);

  expect(apiResponse.ok()).toBeTruthy();

  expect(apiResponse.status()).toBe(200);

});
```

47. What is Visual Testing? Why do we need it?

- Visual Testing is also known as visual comparisons, where two screenshots will be compared. The first screenshot is called the reference or base image, the subsequent run will compare the recent screenshot with reference images and produce the results.
- Visual comparison testing is helpful for UI testing. Using functional testing we will not be able to validate the fonts, styles, typography, alignment, etc. but using the visual comparison we can validate everything related to the application User interface.

48. Write a simple code to Test Visually

For example, if we need to compare the home page we need to write the below code in Playwright.

```
test('Visual test homepage', async ({ page }) => {

  await page.goto('https://playwright.dev');

  await expect(page).toHaveScreenshot();

});
```

- During the first run, the playwright stores the reference image of the homepage, and the next run will be compared against the reference image.
- Optionally we can pass the pixel differences if we need to ignore the minor differences in the image.

```
test('example test', async ({ page }) => {

  await page.goto('https://playwright.dev');

  await expect(page).toHaveScreenshot({ maxDiffPixels: 100 });

});
```

49. How to configure multiple reporters in Playwright?

The playwright allows configuring multiple reporters. The reporter option is available on the playwright.config.js, you can specify the reporter types to configure multiple reporters.

Example:

```
// playwright.config.js

const config = {

  reporter: [

    ['list'],

    ['line'],

    ['json', {  outputFile: 'test-results.json' }]

  ],
};

module.exports = config;
```

50. What is the serial mode in Playwright?

In some scenarios, tests may be inter dependent. The second test might need the output of the first one. Running tests parallelly in such cases will create the test cases to fail and it's like a false failure. The serial mode allows running the tests serially one after the another. If one test fails all remaining tests are skipped and can be retried as a group again.

Example:

```
test.describe.configure({ mode: 'serial' });

let page;

test.beforeAll(async ({ browser }) => {

  page = await browser.newPage();

});

test.afterAll(async () => {

  await page.close();

});

test('runs first', async () => {

  await page.goto('https://playwright.dev/');

});
```

```
test('runs second', async () => {

  await page.getByText('Get Started').click();

});
```

51. How to perform parallel execution in PLaywright?

- The playwright supports parallel execution. Parallel execution can be achieved at the global level or test level in the playwright.
- Parallel in test file level
- The mode: 'parallel' can be passed to describe.configure() function to achieve parallelism.

Example:

```
test.describe.configure({ mode: 'parallel' });

test('runs in parallel 1', async ({ page }) => { /* ... */ });

test('runs in parallel 2', async ({ page }) => { /* ... */ });
```

- Parallel option in the playwright config file
- We can mention the fullyParallel option in the configuration file, this makes the tests run parallelly to all tests.

```
//playwright.config.js

const config = {

  fullyParallel: true,

};

module.exports = config;
```

52. How to perform mobile device emulation in Playwright?

- The emulation features allow testing the application in mobile mode or tablet mode by changing the properties of the browser such as screensize, useragent, geolocation etc.
- For example, if we need to test the mobile safari we can specify the option in the playwright config file like below.

```
const config = {
  projects: [
    {
      name: 'Mobile Safari',
      use: {
        ...devices['iPhone 12'],
      },
    },
  ],
};
module.exports = config;
```

- Similarly, we can set the viewport to match the mobile or tablet screen size

```
const config = {
  use: {
    viewport: { width: 580, height: 720 },

  },

};

module.exports = config;
```

53. Mention some of the helpful ways to debug Playwright tests.

- The playwright provides multiple ways to debug.
- Using the debug option in the command line.

```
npx playwright test --debug
```

- **Debug single test**

```
npx playwright test example.spec.ts --debug
```

VSCode extension

- Apart from the command line debugging option Playwright also provides the VSCode extension "**Playwright Test for VSCode"**
- **Trace on option**
- You can also force the Playwright to record the trace by passing the --trace on option.
- Example:

```
npx playwright test --trace on
```

Pause option

page.pause() can also be used inside the test script to pause the test and do some debugging.

54. What is actionability in Playwright? Explain in detail

- Playwright architecture has a special type of checks before performing any actions on elements. The checks are called actionability checks.
- For example when you do click operation page.click()
- It will perform many checks internally such as
- Element is attached to DOM
- Element is Visible
- Element is Stable and animation is completed(if any)
- Element is ready to receive the events
- Element is enabled.

This mechanism is also called automatic waiting in the Playwright. Since the Playwright performs all the above checks by default one is no need to perform the above checks manually.

55. Mention some of the advantages of Playwright compared to Cypress

- The Cypress and Playwright share a lot of similarities and Playwright overcomes a lot of limitations that cypress has