1) How is Playwright different from other browser automation tools like Selenium?

- Playwright framework supports auto waits, unlike selenium.
- The playwright's execution speed is faster than the execution speed of Selenium.
- It provides support for native browser automation. Additionally, it provides built-in support for taking screenshots and screen recordings while executing tests.
- It allows for cross-browser testing with Chromium, Firefox, and WebKit.
- Provides Better isolation through browser contexts.
- Provides Trace viewer for debugging test failures.

2) What are the supported browsers in Playwright?

Microsoft Playwright supports Chromium, Firefox, and WebKit browsers.

3) How do you install Playwright?

To install Playwright node installation is required. Use the node package manager to install the Playwright package.

```
npm init playwright@latest
```

4) What are the Key features of the Playwright test automation tool?

1. Playwright offers cross-browser support including Chromium, Firefox, and WebKit.
2. The playwright tool can seamlessly execute scripts in both headless and UI modes.
3. Microsoft Playwright supports the auto-waiting feature that automatically waits for elements to be ready before interacting with them. This reduces the need for explicit waits in test scripts.

4. Microsoft Playwright provides built-in capabilities to capture screenshots, videos, and traces that help in debugging.
5. Playwright setup is very easy thus saving significant time and effort.
6. Playwright allows the interception and modification of network requests, enabling advanced testing scenarios where developers can simulate various network conditions.

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

Playwright supports Functional testing, end-to-end testing and API testing.

6) What is a Configuration File in Playwright Framework?

In Playwright, a configuration file manages and customizes the behaviour of Playwright scripts. The configuration file contains several settings and options to configure execution environments, timeout duration, required browsers, screenshots configuration etc.

7) What are some common locator strategies used in Playwright?

The playwright supports the below-mentioned locators:

1. By Id
2. By Text
3. By Text Content

```
page.locator({ text: 'exact text' })
```

4. Xpath

5.CSS selector

6. By Role

```
page.getByRole('button', { name: submit })
```

7. By label

8. By Placeholder

9. By AltText

```
page.getByAltText('AQH')
```

10. By Title

```
//HTML
<span title='AutomationQAHub'>AQH Learning</span>

//Playwright Locator
await expect(page.getByTitle('AutomationQAHub')).toHaveText('AQH Learning');
```

8) What are Playwright Selectors?

Selectors are strings or parameters used to create Locators. Playwright supports selectors like CSS selectors, XPath Selectors, and Text Selectors.

```
//selector
 page.locator('css=button'). //CSS Selector
 page.locator('//div[@id="confirmation"]') //Xpath Selector
```

9) What languages does the Playwright tool support?

Playwright supports JavaScript, TypeScript, Java, C# and Python.

10) How do you record and play scripts in Playwright?

Playwright comes with a test recorder tool named ‘Playwright Codegen’. By using this recorder we can record scripts with basic assertions and use it in our test scripts. To launch the codegen tool execute below mentioned command.

```
npx playwright codegen URL
```

11) What is the difference between fill() and type()?

The Fill() method clears the existing value and then enters the string in one go, In contrast, the type() method just enters the string one by one character.

12) What is the default timeout for the playwright page?

By default timeout for each Playwright test is 30 seconds. If an action is not complete within 30 seconds, the Playwright will throw a timeout error. However, this default timeout can be overridden by providing a custom timeout value when calling specific methods.

```
await page.click('button#submit', { timeout: 5000 });
```

13) What is the default timeout for assertions in Playwright?

For Playwright assertions, the default timeout is 5 Seconds. We can override this default timeout by setting global timeout in the playwright configuration file.

```
 expect : {
    timeout: 30 * 1000,
  }
```

Additionally, we have the option to configure the timeout for each test separately.

```
await expect(navigationMenu.CollapseMenu).toBeVisible({timeout:5000})
```

14) How do you verify a URL in Playwright?

1. Verify for an exact match using expect assertion with the toHaveURL() function.

```
await expect(page).toHaveURL('https://www.automationqahub.com/playwright');
```

2. Verify for partial match

```
await expect(page).toHaveURL(/playwright/);
```

3. Verify for Regular Expression

```
await expect(page).toHaveURL(new RegExp('/playwright$'));
```

15) How to perform click actions in Playwright?

1. To perform generic click

```
page.locator('locator').click()
```

2. To perform a double click

```
page.locator('locator').dblclick()
```

3. To perform right click

```
page.locator('locator').click(button='right')
```

16) What are soft assertions?

Unlike hard assertions, soft assertions do not immediately terminate the execution of the test script. The test execution continues even in the case of assertion failure. The test will be marked as a failure at the end of the execution.

```
  await expect.soft(firstname).toHaveValue('AQH');
```

To collect those soft assertion failures, in the end, we can use a **test. info().errors** to check for the absence/presence of the errors.

```
 expect(test.info().errors).toHaveLength(0)
```

17) What is the difference between innerText() and TextContent() in Playwright?

The innerText() method can fetch visible text of any web element while TextContent() method can retrieve hidden text also.

18) What is the difference between locator() and locateAll()?

Unlike Selenium, Playwright has a single method locator('locator') that is used to find and return the first matching element in the DOM.

```
page.locator('locator')
```

2) Find all matching elements in the DOM and return a list of elements.

```
page.locator('locator').all()
```

19) What are the different commands used to select a dropdown list using Playwright?

In Playwright, three options are available to select the dropdown list.

```
await page.selectOption('select#dropdownId', { label: 'OptionText' });
//SelectByVisisbleText

await page.selectOption('select#dropdownId', { index: 1 }); //SelectByIndex

await page.selectOption('select#dropdownId', 'OptionValue'); //SelectByValue
```

20) How to locate an element using compound selectors in the Playwright Framework?

We can use the OR condition to check the web element that matches either locator1 or locator2.

```
await
expect.soft(page.locator('locator1').or(page.locator('locator2'))).toContainT
ext(team);
```

Another approach is by using a comma operator.

```
await expect.soft(page.locator('button:has-text("Log in"), button:has-
text("Sign in")')).toBeVisible();
```

21) What is the purpose of the waitForFunction method in Playwright, and how is it used?

waitForFunction() is used to pause execution until a given function returns a truthy value. For example: await page.waitForFunction(() => window.innerWidth < 1000);

22) Explain WaitFor() in Playwright.

WaitFor() is a function in Playwright that is used to wait for specific conditions like visible, stable, enable and attached.

23) How to perform scrolling in Playwright?

```
 await page.evaluate(() => { window.scrollTo(0, document.body.scrollHeight);
});
await page.locator('locator').scroll_into_view_if_needed()
```

24) How to retry failed test cases in Playwright?

We can change the configuration settings in the **playwright.config.js** file.

```
const config = {
  // Give failing tests 2 retry attempts
  retries: 2,
};
```

An alternative option is to provide a retry mechanism from the command line.

```
npx playwright test --retries=2
```

25) Name a few Playwright exceptions.

Some of the common exceptions are listed below:

1. **Timeout Error**: This exception is thrown when a specific operation times out. For example, waiting for an element to appear on the page using waitForSelector might throw a TimeoutError if the element doesn’t appear within the specified timeout.
2. **ElementHandleError**: This exception is thrown when there is an issue with interacting with an element handle. This could happen if the element is not visible, not clickable, or if the operation cannot be performed.
3. **NetworkError**: Network Error is thrown for network-related errors, such as failed network requests or inability to connect to a resource.
4. **ContextClosedError**: This error occurs when operating on a closed browser context.
5. **StrictModeViolation Error**: This error occurs when our locator finds more than one element.

26) How to save a screenshot to the path?

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

This will take a screenshot of the page and store it at the project level.

27) How to take partial screenshots with the Playwright?

To take a partial screenshot or a web-element-specific screenshot use locator.screenshot() method.

```
await page.locator('#textbox').screenshot({path:'PartialScreenshot.png'})
```

28) What is Browser Context in Playwright?

"browser context" refers to an isolated environment within a browser instance. A browser context helps in creating multiple, independent instances of a browser, each with its own set of pages, cookies, and browser storage. This is useful for managing different states or scenarios, parallel execution and network isolation.

```
 // Launch a browser instance
  const browser = await chromium.launch();

  // Create two browser contexts
  const context1 = await browser.newContext();
  const context2 = await browser.newContext();

  // Create pages in each context
  const page1 = await context1.newPage();
  const page2 = await context2.newPage();

  // Navigate to different URLs in each page
  await page1.goto('https://example.com');
  await page2.goto('https://example.org');

})();
```

We created 2 browser contexts and different pages within each context. Both pages open different URLs. We can perform different sets of actions in these contexts.

29) How to automate alert popups in Playwright?

In Playwright popup alerts can be automated using a **page.on('dialog')** event handler. This event handler allows you to intercept and handle various types of dialogues, including alert, confirm, and prompt dialogues.

```
page.on(dialog,dialog =>dialog.accept()) //To Accept the alert
page.on(dialog,dialog =>dialog.dismiss()) //To Dismiss the alert
```

30) How do I run tests parallel in the Playwright framework?

By default, Playwright executes test files in parallel mode, yet it processes test cases within these files sequentially. For instance, if you have 3 test files, each containing 2 test cases, all 3 test files will execute in parallel mode, while the tests within each file will run sequentially.

31) How to execute Playwright tests in parallel mode from a single test file?

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

This code will instruct to execute all the tests in the test file parallel.

32) How to execute Playwright tests in Serial mode from a single test file?

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

This is the default setting in each test file. We can override it with 'parallel' mode to execute test cases in parallel mode.

33) How to download a file using Playwright.

The playwright provides a page.waitForEvent('download') event handler to support download events.

```
 await page.locator('locator').click()
 const[download] =await Promise.all([
    await page.waitForEvent('download')
```

```
  ])
  console.log(`Download started: ${download.suggestedFilename()}`);
  const path =download.suggestedFilename();
  await download.saveAs(path)
```

Upon Clicking on the link/Button, a download event is triggered which is handled using the page.waitForEvent('download') event handler. Once the download event is received, we save the file to a specified path and log the information.

34) How to attach a file in the HTML report?

TestInfo contains information about the currently running test. It is available to test functions, hooks and test-scoped fixtures.TestInfo provides utilities to control test execution: attach files, update test timeout, determine which test is currently running and whether it was retried, etc. To attach the downloaded file to the test report use the below code.

```
await testInfo.attach('downloaded file', { path: path });
```

35) How to generate and share the Allure report/HTML report in the Playwright framework?

In this article, I have explained in detail how HTML/Allure reports can be generated and shared by compressing the test results in the Playwright test automation framework.

36) How to Handle alerts in Microsoft Playwright?

```
 page.on('dialog', async dialog => {
           // Verify type of dialog
  expect(dialog.type()).toContain('alert')
  });
```

The above code verifies that the alert is present on the dom. To verify the text of the alert box write the below code.

```
  // verify message of alert
  expect(dialog.message()).toContain('Please select atleast one pillar
phase.');
```
To accept the alert use below code snippet.

```
 //click on alert ok button
  await dialog.accept();
```
37) Which is better Playwright or Cypress?

Every tool has pros and cons. Tool selection also depends on the project requirements and constraints. However follow this article to understand the differences between Cypress, Playwright and Selenium.

38) Can Playwright automate a Windows application?

Microsoft Playwright can automate Electron-based desktop applications across various platforms, including Windows, MacOS, and Linux with the help of a package nut js. This is applicable with JavaScript only.

39) How to upload a file in Playwright?

setInputFile() command is used in Playwright to upload a file. If we want to upload multiple files then we can use the setInputFiles() method.

```
await page.locator('#Button').setInputFiles('myfile.pdf');
await page.locator('#Button').setInputFiles(['file1.pdf', 'file2.pdf']);
```
40) How to close a page/tab or browser in Playwright?

```
//To Close Page or Tab
page.close() - Used to close individual page

// To close Browser
browser.close()
```