CUCUMBER INTERVIEW QUESTIONS

What is Cucumber?

- In real time projects, whenever automations scripts are developed, you need to create an **execution system** which is called a framework to **run** and **maintain automated tests.**
  - Cucumber is a BDD framework. It does NOT automate your test cases and it is NOT a system to write a test case BUT to **maintain** and **execute** them.
  - Since it is an execution system, it can run any test; web, mobile, API, unit testing which is written in **JAVA/RUBY.**
  - One of its wonderful main features is the ability to execute plain text functional description (written in language named Gherkin) as automated tests.
  - Thanks to the Gherkin language, Cucumber is good for the non-technical people as it is easy to read.
  - Cucumber works with **JUnit** and **TestNG**. In my company, we use it with JUnit.
  - You can achieve data driving testing, parameterization, get reports, control executions, use hooks with Cucumber.
  - **In my project:**
    We use cucumber for our automation testing. Every sprint I get new stories from **Jira** and write feature files for them. Then I automated those feature files. I can also run my smoke and regression tests with cucumber.

What makes Cucumber unique?

- Gherkin Language
- Features file
- Step Definitions
- Runner Classes
- Hook Class
- Tags

What are the advantages of BDD?

- **Readability**: Clear requirements written with Gherkin language which is plain English
- **Reusability**: You don't have to write a new code for the same functions thanks to how steps_definitions are created based on the scenario steps in the feature files.
- **DDT**: Data Driven testing is very easy with Scenario Outlines and Examples Table.
- Quick and easy set up and execution
- Efficient tool for testing

What is Gherkin Language?

- It is a Business Readable Domain Specific Language that lets you describe software's behaviour.
- Thanks to Gherkin, we use more simple language which allows non-technical people to understand the codes.
- Gherkin is used by feature files.

- The Keywords are Feature, Scenario, Given, Then, When, And, But, BackGround, Scenario Outline.

What does "@Cucumber Options" do?

```
@CucumberOptions(
    plugin = { //plugin: It is used to configure what reports we want to generate and where to put them.
        "json:target/cucumber.json",
        "html:target/default-cucumber-reports", "rerun:target/rerun.txt"
                                        },
    tags = {" @storemanager"}, //tags: you can specify the tests you would like to run here.
    features = {"src/test/resources/features" //to specify the path of the feature files
    },  //feature contains scenarios; every scenario is like a test; where is the implementation for features
    glue = {"com/vytrack/step_definitions"}, // this is where to look for step definitions. Hook is part of glue as well
    dryRun = false //dry run - to generate step definitions automatically if it is true. You will see them in
the console output      )
```

What are the components of Cucumber BDD Framework?

- **Feature File:**
  This is where we define the scenarios with plain and simple English which test a certain functionality or feature of the application. Feature is the main story while **scenarios** are test cases.

```
@vehicles
Feature:
Vehicles
  As user I want to see list of all vehicles
  Scenario: Login as driver and navigate to the Vehicles
        Given user is on the landing page
        When user logs in as a "driver"
        Then user navigates to "Fleet" and "Vehicles"
        And user verifies that "Cars" page name is displayed
        And user verifies that default page number is 1
```

- **Step Definitions:**
  This is where we write the codes and implement automation for the scenarios in the feature files. Step definitions are in the same package with cukesRunner, or child package (not parent or sibling).

```
@Given("user is on the landing page")
public void
user_is_on_the_landing_page() {
pages.loginPage().goToLandingPage();
}

@Then("user logs in as a store manager")
public void
user_logs_in_as_a_store_manager() {
  String username =
```

```
        ConfigurationReader.getProperty("storemanagerusername");        String
        password = ConfigurationReader.getProperty("storemanagerpassword");
        pages.loginPage().login(username, password);
    }
```

- **Cukes Runner:**
  This is where we run all of the tests with JUnit. This class includes @CucumberOptions which includes plugin, tags, features, glue, etc.

```
@RunWith(Cucumber.class) // this is from JUnit
@CucumberOptions(
    plugin = {
        "json:target/cucumber.json",
        "html:target/default-cucumber-rep
        orts", "rerun:target/rerun.txt"     },
    tags = {" @storemanager"},
    features = {"src/test/resources/features"
                        }, glue =
    {"com/vytrack/step_definitions"},
    dryRun = false       )
```

How does the .FEATURE FILE work?

Feature: *description of what is being tested @tags. Sample feature file;*

Feature : login functionality → Background:
Given : I am on the login page → Scenario: 1, Scenario: 2
*The **background** runs before both of the scenarios*

Scenario: *description of the scenario being test*

Given I am on the login page
And I enter username and password
When I click on the submit button
Then I should be able to see the profile picture
But the submit button should not be displayed

Given: a precondition
When: condition that triggers the expected result
Then: expected condition

How do you execute / run Cucumber tests?

1. Create a feature file
2. Have a tag:
   not required but you should have if there are many test cases
3. Generate Step definitions and write your test codes in there
4. Configure Cukes Runner:
   a. @RunWith(Cucumber.class)
   b. glue
   c. features
   d. Tags
5. Run as JUnit

Cucumber Workflow

Cucumber works by executing a specific method known as **step_definition** which matches the **steps from the feature file**. Step definition methods will use page object models, utilities etc.

Feature —> step definitions —> Page object model

What is Background?

- Background helps us to define a step or series of steps that are **common to all of the tests in the feature file**.
- Background runs **before each and every scenarios** in the feature file.
- Background can only put **on the top** (first in the feature file), before all scenarios
- You cannot put pipelines in backgrounds (Only in scenario outline)
- Background and **Hooks** cannot be used at the same time. Background is only .feature file level while Hooks are global to all tests.
- How did you use Background in your project?

What are Hooks in Cucumber?

- Cucumber supports hooks which are blocks of code that run **before** and **after each scenario (test case)**.
- Cucumber hook allows us to **better manage the code workflow** and helps us to **reduce the code redundancy**. We can say that it is an unseen step, which allows us to perform our scenarios or tests.
- Hooks are very similar to @BeforeMethod & @AfterMethod in testNG.
- **Hook** Class must be located under the step_definitions **package** or *if you want to have it under a different package, you need to specify inside a runner class glue for the package with hook*.
- Hook Class will not run if dryRun=true
- In my project, I implement **screenshots** inside Hook Class.
- I use Scenario as a parameter in my before/after method
- I use **@Before** for the actions I want to be implemented by any scenarios in the project.
- Actions: maximizing the window, setting the implicit wait, and starting the webdriver.

```
@Before
public void setup(Scenario scenario){
Driver.getDriver().manage().window().maximize();
Driver.getDriver().manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);
Driver.getDriver().get(ConfigurationReader.getProperty("url" +
ConfigurationReader.getProperty("environment"))); }
```

Screenshot in Cucumber in Hooks Class

- I have a Hook Class in my project and in the @After method I use the following codes to take screenshots of failed tests in my project.

```
@After
public void teardown(Scenario scenario){
  if(scenario.isFailed()){
    TakesScreenshot takesScreenshot = (TakesScreenshot) Driver.getDriver();
    byte[] image = takesScreenshot.getScreenshotAs(OutputType.BYTES);
    //will attach screenshot into report
    scenario.embed(image, "image/png");
  }
  Driver.closeDriver();
  System.out.println("AFTER");
}
```

How can you run a hook only for specific scenarios?

- @Before("@storemanager")
- This will make @Before hook run for the scenarios (test cases) with @storemanager. Otherwise, the default hook will run always.

How to change the priority in hooks?

- @Before(order=2)
- @Before(value = "@Storemanager", oder=1)

Why do you need different hooks?

- If there are only 1-2 features which require a special or specific set-up, we may want to have a separate hook for it. For example, if you need to use **DataBase** for some of the features, you may want to create a hook for those DataBase required features.

What is DataTable in Cucumber?

- Cucumber DataTable allows us to overcome issues with a big set of data or even more we cannot fit all data into steps. Thanks to DataTable, we don't have to have many same steps but have one step for all sets of data.
- DataTable comes from Cucumber and based on our need we can choose from one of the following data storing options: **E, List<E>, List<List<E>>, List<Map<K,V>>, Map<K,V>**
- If you have one **column** as in the example below, this is passed as a **List<String>** to the step definitions method.

```
Scenario: Verify column names
Given user logs in as a store manager
When user navigates to "Activities" and "Calendar Events" Then following table headers should be displayed
| TITLE |
| CALENDAR |
| START |
| END |
| RECURRENT |
| RECURRENCE |
```

- In the example below, we have two columns that are stored as **Map <String, String>**. There are only 2 columns for MAP; Key on the left, Value on the right.

```
| Key Name | Value |
| UserName | Storemanager21 |
| PassWord | useruser35 |
```

- When there is a real table with multiple rows and columns, we use **List<Map<String,String>>** inside the step_definition. This is basically map objects stored into a list. This is how I use **Maps** in my framework without Scenario Outline!

```
...
And user enters car information to create a car
| License Plate | Driver | Location | Year | Color | Power | Make | Model |
| Cybertek | Spartan | Alaska | 2019 | Black | 500 | BMW | X5M |
| BobX304 | Fearlesi | Indiana | 2020 | Black | 300 | Honda | Civic |
Then user clicks save and close
And user verifies that general information is displayed
```

Scenario Outline and Parameterization in Cucumber

- If you want to run your test case multiple times with different data sets, we call it **parametrization**. We may also say **Data Driven Testing** which allows us to automatically run a test case multiple times with different inputs and validation values.
- Scenario in cucumber runs only once while scenario outline executes multiple times with the given datasets.
- To achieve parameterization in cucumber, you need to write Scenario Outline in your .feature file. This is how I use **Maps** in my framework with Scenario Outline!

```
@login_1
Scenario Outline: login as different user tests
 Given user logs in as a "<user_type>"
 And user verifies that "<page name>" page name is displayed
      Examples:
       | user_type      | page name      |
       | driver         | Quick Launchpad |
       | sales manager  | Dashboard      |
       | store manager  | Dashboard      |
```

Data Driven Testing

- Data-driven testing is a test automation framework which stores test data in a table or spreadsheet format. This allows us to have a single test script which can execute tests for all the test data in the table.
- In this framework, input values are read from data files and are stored into a variable in the test script. DDT enables building both positive and negative test cases into a single test.
- Test data is separated from code and stored into external sources: **Cucumber Examples table, Excel files, CSV files, Database**.
- If the amount of data is not that huge, then I use the Cucumber Scenario outline with the Examples table.
- And other times I maintain test data in Excel files, and I use Apache POI library to read and write data.
- If data comes from a database, or I need to do database validation, I use SQL queries along with JDBC library in java.

How do I use the cucumber scenario for DDT?

- In my current project I use Scenario Outline with Examples @login_1

```
Scenario Outline: login as different user tests
 Given user logs in as a "<user_type>"
 And user verifies that "<page name>" page name is displayed
 Examples:
  | user_type      | page name      |
  | driver         | Quick Launchpad |
```

| sales manager | Dashboard |
| store manager | Dashboard |

How do you do DDT with Cucumber?

1. Use **Scenario Outline** that allows to run the same test with different DataSets.

2. Use **DataTables** which allow you to store test data under the test step in the feature file.

How to run ONLY failed tests in Cucumber?

- Cucumber can generate **txt with failed scenarios.** This file helps us to rerun the failed tests stored in it.
- You may want to have a separate second Runner class only for failed scenarios and configure Cukes Runner in it accordingly.
- Add the second runner in the pom file

FailedRunner Class:

```
@RunWith(Cucumber.class
) @CucumberOptions(
    plugin = {
        "json:target/cucumber.json",
        "html:target/default-cucumber-rep
        orts",
"rerun:target/rerun.txt" failed scenarios are stored in this txt file. We have this in the
                         CukesRunner class as well, so it returns all failed tests/scenarios.
    },
  features = {"@target/rerun.txt" //to specify where txt file with failed features
    },
    glue =
    {"com/vytrack/step_definitions"},
    dryRun = false
```

- To **automate FailedRunner** Class, you need to configure the **POM file** and define the order so the CukesRunners will be executed first and then FailedRunner will be executed.

```
<configuration> //alphabetical order will run the CukesRunner first and FailedRunner second.
    <runOrder>alphabetical</runOrder>
    <includes>
        <include>**/FailedRunner.java</include>
        <include>**/CukesRunner.java</include>
    </includes>
</configuration>
```

How to rerun the failed tests again in Jenkins?

- In Jenkins there are plugins that re-run the failed tests Unit cases.
- So you can configure your **Maven build execution** on Jenkins using the option: **Dsurefire.rerunFailingTestsCount=2**

What reports do you generate in Cucumber?

In my framework, I can generate two different html reports. My reports have detailed steps and the screenshot for failures.

1. **Default cucumber html report:**
   We get the report regardless of whether we run it with Maven or CukesRunner. This is a default report which means we don't need to do any configuration in the pom file. As long as the cucumber runs, we will get the report.

```
@CucumberOptions(
    plugin = {
        "json:target/cucumber.json",
```

```
        "html:target/default-cucumber-re
        ports", "rerun:target/rerun.txt"
    },
```

2. **Maven-cucumber-reporting:** *(This what we will talk in the interview)*
   This is a **plugin** in the pom file so we need to add this in the pom to make it work. We also need to add **JSON** option in the CukesRunner. To generate this plugin, we always have to run using terminal/maven. This report provides more metrics, pass & failed rates, enables sorting by tags, etc.

   In my cucumber framework, I use a third party tool **maven-cucumber-reporting**. This is how it works:

   1. I add the **json** option in my CukesRunners as shown below:

```
@CucumberOptions(
    plugin = {
        "json:target/cucumber.json",
        "html:target/default-cucumber-rep
        orts", "rerun:target/rerun.txt"
    },
```

   2. And then, I add the **maven-cucumber-reporting** to the pom file.

```
<groupId>net.masterthought</groupId>
<artifactId>maven-cucumber-reporting</artifactId>
<version>4.8.0</version>
```

   3. In the configuration of the maven-cucumber-reporting plugin, point to the json from step 1.

```
<configuration>
  <projectName>Cucumber HTML Reports</projectName>
  <outputDirectory>${project.build.directory}</outputDirectory>
                    this will specify the location of the report.
```

   4. Run as maven command *mvn verify* to execute the tests and generate reports. This report will not generate if we do not run it this way.

```
<execution>
  <id>execution</id>
  <phase>verify</phase>
  <goals>
```

How to achieve 100% test coverage through framework?

- In cucumber, we are committed and must execute each and every line in the feature file. **Gherkin** language helps us or we can say it mandates us to do so. If we do not execute any line, we will see it as an error in the output saying that specific step definition was not implemented so we have to write a code block for it.

How do you run only certain group features and certain tests (scenarios)?

- Tags feature will help us to tag the steps and run them from the runner file (class)

- In the runner class, you can specify the path to the folder which keeps files for the feature file that you would like to run / execute.

```
features = {    "src/test/resources/features/fleet"
                "src/test/resources/features/login"
        },                                  only "fleet" and "login" features will run
```

- In my **.feature files**, I use **tags** and pass them to the **CukesRunner**.

```
tags = {" @storemanager"}, this will help me run "@Smoke" tests only
```

- Tags and features can also be passed using the command line

mvn test -Dcucumber.options="--tag @smoke"

How do you write and maintain reusable code across the framework?

- All of the **feature file scenarios** are automatically pointing to the **step definitions code**. If there is a specific scenario step repeating in the different scenario, there will be only one code block for them in the step definitions so we don't repeat our code. Step definitions and parameterization in cucumber help us a lot!

Cucumber + Jira + Xray

- In my project, my tests in **Jenkins** are connected to **Jira.** We achieve this with a special plugin, **Xray** in Jira. The results of my smoke tests will be reported to Jira everyday.

How to run Cucumber with JUnit?

- Add cucumber JUnit dependency
- Adding @RunWith (Cucumber.class) on top of cukesRunner class

What happens if you run your runner class with no tags?

- All the feature files will run from top to bottom but only the feature files that are located in the @CucumberOptions "features="

Framework Tools : Cucumber BDD framework

- Junit, Cucumber Java, Maven
- Selenium, HTML reporting with screenshots Log4j,
- JDBC, Rest Assured, Apache POI, Git, Jenkins

Page Class in Utilities package

- Pages class works like a factory for page objects. Through **Page Instance**, we can access any page object/instance since all of the page will be like a child class of the Pages class. This way, we do not have to call each page or we do not have to create an object of the pages to reach their methods BUT we can create only one Pages object and reach all other pages starting from there.

```
Pages pages = new Pages();
```

If we need anything from LoginPage, we don't have to create any other object for LoginPage since we can reach it from Pages object now. See the code below:

```
public class Pages {
  private LoginPage loginPage;
  private CalendarEventsPage calendarEventsPage;
  private DashboardPage dashboardPage;
  private ManageDashboards manageDashboards;
  private VehiclesPage vehiclesPage;

  public VehiclesPage vehiclesPage() {
    if (vehiclesPage == null) {
      vehiclesPage = new VehiclesPage();     }
    return vehiclesPage;      }

  public LoginPage loginPage() {
    if (loginPage == null) {
      loginPage = new LoginPage();     }
    return loginPage;}
```