Cucumber:

Cucumber is a behaviour-driven development (BDD) testing tool. It provides a method for writing tests that anyone, regardless of technical knowledge, can comprehend. We can write test scripts from both the developer's and the customer's perspectives with Behavior Driven Development.

The Ruby programming language was originally used in the creation of the Cucumber framework. Cucumber now supports a variety of programming languages, including **Java** and **JavaScript**, through several implementations. SpecFlow is the name of the open-source Cucumber port for .Net.

Certainly! Here are some basic commands and concepts for working with Cucumber:

1. **Create a Feature File:** Feature files are written in Gherkin syntax and contain scenarios describing the behavior of your application. Create a `.feature` file and write your scenarios using Given-When-Then steps.
2. **Run Cucumber Tests:**
    - To run Cucumber tests, you typically execute the test runner class which will locate and execute your feature files.
    - In Java, you can use the JUnit or TestNG test runners. In Ruby, you might use RSpec.

- Use the appropriate test runner command depending on your programming language and test framework.

3. **Step Definitions:** Step definitions map the Given-When-Then steps in your feature files to the actual code that should be executed. Each step in the feature file corresponds to a step definition method.
4. **Regular Expressions:** Step definitions are matched to steps in feature files using regular expressions. Regular expressions define patterns that match step text.
5. **Hooks:** Hooks are blocks of code that run before or after each scenario. They can be used for setup and teardown tasks such as opening a browser before each scenario and closing it afterward.
6. **Tags:** Tags allow you to categorize your scenarios. You can run scenarios with specific tags using the test runner's tag option.
7. **Data Tables and Scenario Outline:** Data tables and scenario outlines allow you to parameterize your scenarios and run them with different sets of data.

For theory:

1. **Gherkin syntax**: Ensure you have a strong grasp of writing clear and concise feature files using Gherkin syntax, including scenarios, scenario outlines, and tags.
2. **Step definitions**: Understand how step definitions are written in your programming language of choice (e.g., Java, Ruby) and how they interact with Cucumber to execute scenarios.
3. **Hooks**: Learn about hooks in Cucumber and how they can be used to set up preconditions or clean up after scenarios.
4. **Data tables and scenario outlines**: Explore how to use data tables and scenario outlines effectively to parameterize your scenarios and make them more reusable.
5. **Tags and filtering**: Understand how to use tags to organize and filter your scenarios, allowing for selective test execution.
6. **Best practices**: Familiarize yourself with best practices for writing maintainable and efficient Cucumber tests, including guidelines on feature file structure, step definition organization, and scenario design.

Keywords:

Sure, here's a one-liner description for each keyword along with Hooks:

Structural Keywords:

1. **Feature:** Describes a feature or functionality of the software.
2. **Background:** Defines steps that are common to all scenarios within a feature file.
3. **Rule:** Defines a high-level business rule in the feature file.

Scenario Structure Keywords:

4. **Scenario:** Describes a specific test scenario.
5. **Scenario Outline:** Allows for the same scenario to be executed multiple times with different inputs.
6. **Examples:** Provides different sets of data for a Scenario Outline.

Step Keywords:

7. **Given:** Describes the initial state or preconditions for the scenario.
8. **When:** Describes the action or event that occurs.
9. **Then:** Describes the expected outcome or result of the scenario.
10. **And:** Chains together Given, When, or Then steps.
11. **But:** Specifies an exception or alternative condition to the previous step.

Additional Keywords:

12. **Hooks:** Executes setup and teardown actions before and after scenarios or test events, such as starting and stopping services or setting up test data. Hooks are not directly defined within feature files but are part of the test automation infrastructure.

Software tools supported by Cucumber

The piece of code to be executed for testing may belong to different software tools like **Selenium**, **Ruby on Rails**, etc. But cucumber supports almost all popular software platforms, and this is the reason behind Cucumber's popularity over other frameworks such as **JDave**, **Easyb**, **JBehave**, etc. Some Cucumber supported tools are given below:

- Ruby on Rails
- Selenium
- PicoContainer
- Spring Framework
- Watir

Jsing variables and examples

o, we all know that there are more days in the week than just Sunday and Friday. Let's update our scenario to use variables and evaluate more possibilities. We'll use variables and xamples to evaluate Friday, Sunday, and anything else!

Update the `is_it_friday_yet.feature` file. Notice how we go from `Scenario` to `Scenario Outline` when we start using multiple `Examples`.

```
Feature: Is it Friday yet?
  Everybody wants to know when it's Friday

  Scenario Outline: Today is or is not Friday
    Given today is "<day>"
    When I ask whether it's Friday yet
    Then I should be told "<answer>"

  Examples:
    | day            | answer |
    | Friday         | TGIF   |
    | Sunday         | Nope   |
    | anything else! | Nope   |
```

Hooks:

Hook Annotations

Unlike TestNG Annotations, the cucumber supports only two hooks:

- **@Before**
- **@After**

@Before

As the name suggests, we can use the **@Before** hook with the function/method after which we need to start web driver.

@After

As the name suggests, we can use the **@After** hook with the function/method after which we need to close the web driver.

Let's understand this notion better with an example of a step definition file.

Cucumber Plugin Names:

1. **Cucumber-JVM:** The core implementation of Cucumber for Java Virtual Machine-based languages like Java, Kotlin, and Groovy.
2. **Cucumber-Ruby:** The implementation of Cucumber for Ruby, enabling behavior-driven development in Ruby projects.
3. **Cucumber.js:** Allows writing Cucumber tests in JavaScript for Node.js applications.
4. **Cucumber-Eclipse:** Provides integration with Eclipse IDE for writing and running Cucumber tests in Java.
5. **Cucumber IntelliJ:** Offers integration with IntelliJ IDEA for writing and executing Cucumber tests in Java.
6. **Cucumber Reporting:** Provides enhanced reporting capabilities for Cucumber test results, often used for generating HTML reports.
7. **Cucumber Extent Report:** Integrates Extent Reports with Cucumber to generate rich and interactive HTML reports for test execution results.
8. **Cucumber TestNG:** Integrates Cucumber with TestNG, allowing Cucumber tests to be executed using TestNG's testing framework.
9. **Cucumber Spring:** Provides integration between Cucumber and the Spring Framework for Java applications, enabling dependency injection and other Spring features in Cucumber tests.
10. **Cucumber Parallel:** Enables parallel execution of Cucumber tests, improving test execution speed by running scenarios concurrently.

Tag Combinations used in Runner File:

1. **Single Tag:**
   - `@smoke`: Executes scenarios tagged with `@smoke`.
2. **Multiple Tags (Logical AND):**
   - `@regression and @ui`: Executes scenarios tagged with both `@regression` and `@ui`.
   - `@regression and not @wip`: Executes scenarios tagged with `@regression` but not `@wip`.
3. **Multiple Tags (Logical OR):**
   - `@api or @integration`: Executes scenarios tagged with either `@api` or `@integration`.
   - `@slow or @performance`: Executes scenarios tagged with either `@slow` or `@performance`.
4. **Combination of AND and OR:**

- `(@critical or @high) and not @deprecated`: Executes scenarios tagged with either `@critical` or `@high` but not `@deprecated`.

5. **Using Parentheses for Grouping:**
   - `(@sanity or @smoke) and (@ui or @api)`: Executes scenarios tagged with either `@sanity` or `@smoke` and also tagged with either `@ui` or `@api`.
6. **Combining Logical AND, OR, and NOT:**
   - `(@regression or @smoke) and not (@ui or @integration)`: Executes scenarios tagged with either `@regression` or `@smoke` but not tagged with either `@ui` or `@integration`.

Sure, here are the names of some popular Behavior-Driven Development (BDD) tools:

1. Cucumber
2. JBehave
3. RSpec
4. Cucumber-JS
5. Jasmine
6. Behave
7. Lettuce
8. SpecFlow
9. NUnit
10. Behat
11. PHPSpec
12. Gauge
13. FitNesse

Sample code

cucumber.feature | annotation.py

```
1  Feature: Character count
2
3  @Java
4  Scenario: Count characters in usernames
5  Given a set of specific users
6  | name | noOfCharsInName |
7  | John | 4 |
8  | Alice | 5 |
9  | Bob | 3 |
10 When we count the number of characters in "Alice"
11 Then we will find "2" people's with length "5"
12 Then we will find only "1" person with length "9"
```

Scenario outline example

gherkin Copy code

```
Feature: Login functionality

  Scenario Outline: Successful login with valid credentials
    Given the user is on the login page
    When the user enters "<username>" and "<password>"
    And clicks the login button
    Then the user should be redirected to the dashboard page

    Examples:
      | username     | password  |
      | user1        | pass1     |
      | user2        | pass2     |
      | user3        | pass3     |
```

java

Copy code

```java
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import io.cucumber.java.en.*;

public class LoginSteps {
    WebDriver driver;

    @Given("the user is on the login page")
    public void the_user_is_on_the_login_page() {
        driver = new ChromeDriver();
        driver.get("http://example.com/login");
    }

    @When("the user enters {string} and {string}")
    public void the_user_enters_and(String username, String password) {
        driver.findElement(By.id("username")).sendKeys(username);
        driver.findElement(By.id("password")).sendKeys(password);
    }

    @When("clicks the login button")
    public void clicks_the_login_button() {
        driver.findElement(By.id("loginButton")).click();
    }

    @Then("the user should be redirected to the dashboard page")
    public void the_user_should_be_redirected_to_the_dashboard_page() {
        String currentUrl = driver.getCurrentUrl();
        assert(currentUrl.equals("http://example.com/dashboard"));
        driver.quit();
    }
}
```