Behaviour Driven Development (BDD) & Test Automation

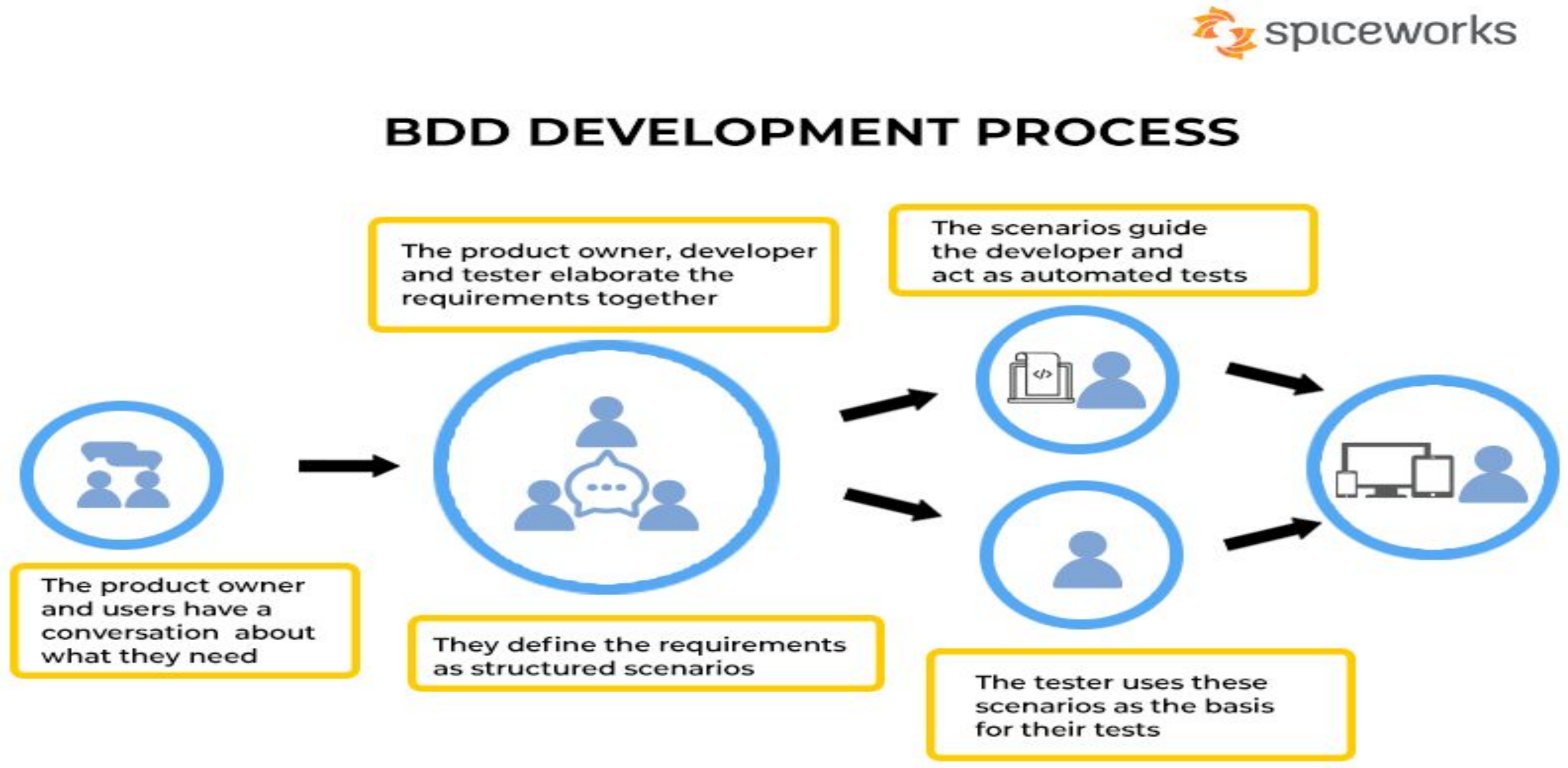
Md.Nahid Hossain

Part-1

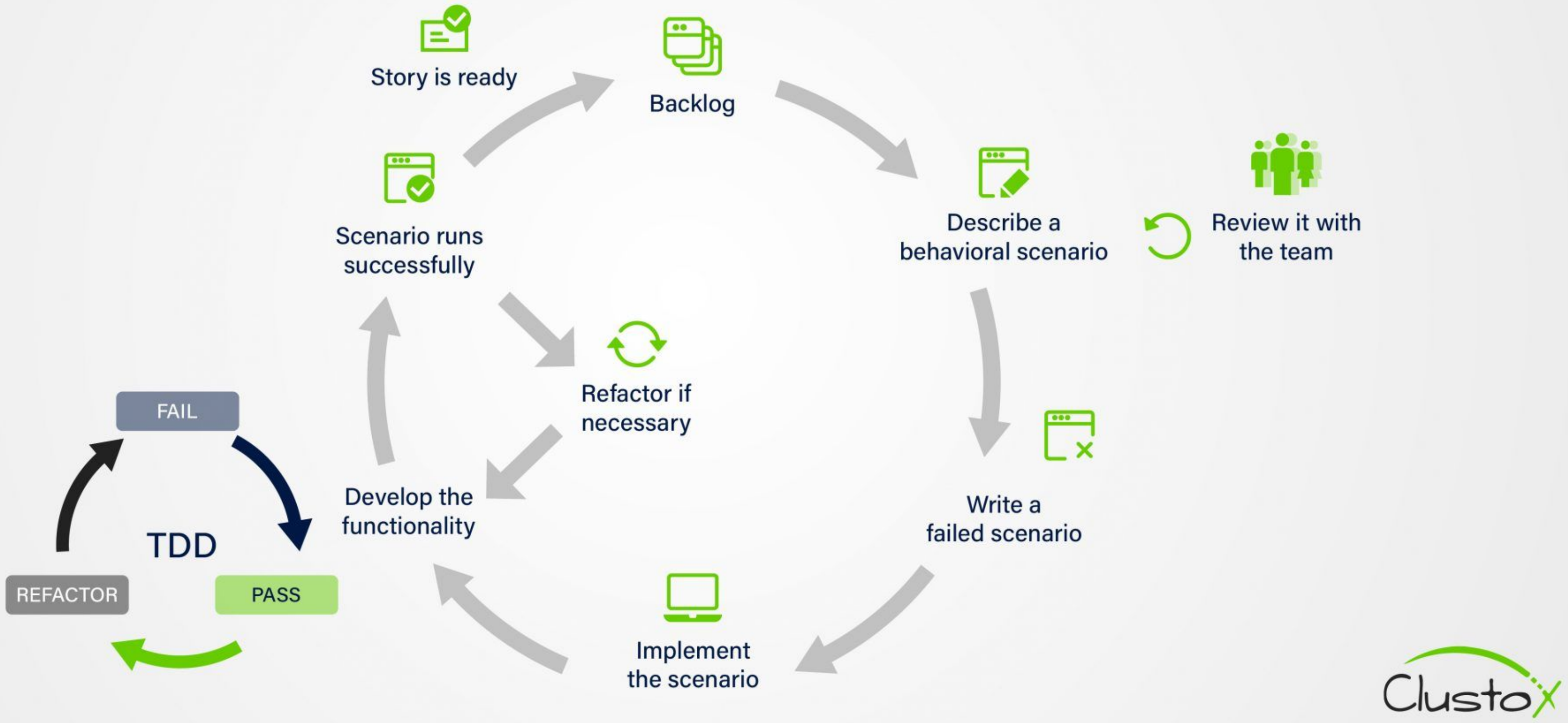
What is BDD?

Behavior-driven development (BDD) is an Agile software development methodology in which an application is documented and designed around the behavior a user expects to experience when interacting with it.

BDD as a process


spiceworks
BDD DEVELOPMENT PROCESS
The product owner and users have a conversation about what they need
The product owner, developer and tester elaborate the requirements together
They define the requirements as structured scenarios
The scenarios guide the developer and act as automated tests
The tester uses these scenarios as the basis for their tests

The BDD Process



Gherkin Format

Feature: Log In
Log in must be fast and user-friendly

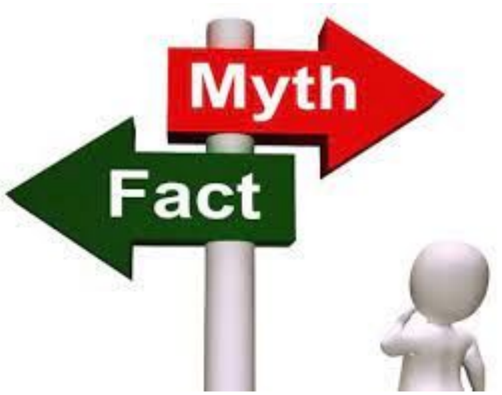
Scenario: Successful Log In
User should get a pop-up message of ‘Signed In’ on screen
Given the user has chosen to sign up
When the user sign up with the valid credentials
Then they should receive a pop up message

Scenario: Invalid Credentials
When a user tries to log in using the credentials which don’t exist.
Scenario: Invalid Password
When a user puts invalid password

This helps visualize the whole system feature and its behavior under different conditions. It also aids the collaborative approach taken in behavior-driven development as it is easily understandable by various teams, businesses, and stakeholders.

Cucumber

“Cucumber is a tool that supports Behaviour-Driven Development(BDD)”



“Myth: Using Cucumber means you’re doing BDD”

Scenario: User Registration

Define the Feature: Start by identifying the feature or functionality you want to develop. In this case, let's focus on user registration.

Write the Feature Description: Describe the feature in a user-centric manner. For example, "As a user, I want to be able to register on the website so that I can access exclusive content."

Create the Feature File: Create a feature file using a BDD framework such as Cucumber, SpecFlow, or Behat. The feature file should be written in a format that is readable by both technical and non-technical stakeholders.

Example : User Registration

```
Feature: User Registration
  Scenario: Successful user registration
    Given I am on the registration page
    When I fill in the registration form with valid details
    And I click the "Register" button
    Then I should be redirected to the login page
    And I should receive a confirmation email
```

Acceptance Criteria

Step 3: Write Step Definition

```
Given("I am on the registration page", () -> {
  // Code to navigate to the registration page
});

When("I fill in the registration form with valid details", () -> {
  // Code to fill in the registration form with valid data
});

When("I click the {string} button", (String button) -> {
  // Code to click the specified button
});

Then("I should be redirected to the login page", () -> {
  // Code to verify the user is redirected to the login page
});

Then("I should receive a confirmation email", () -> {
  // Code to check if a confirmation email is received
});
```

Execute the Tests: Run the BDD tests using the BDD framework. The framework will execute the steps defined in the feature file and match them with the corresponding step definitions.

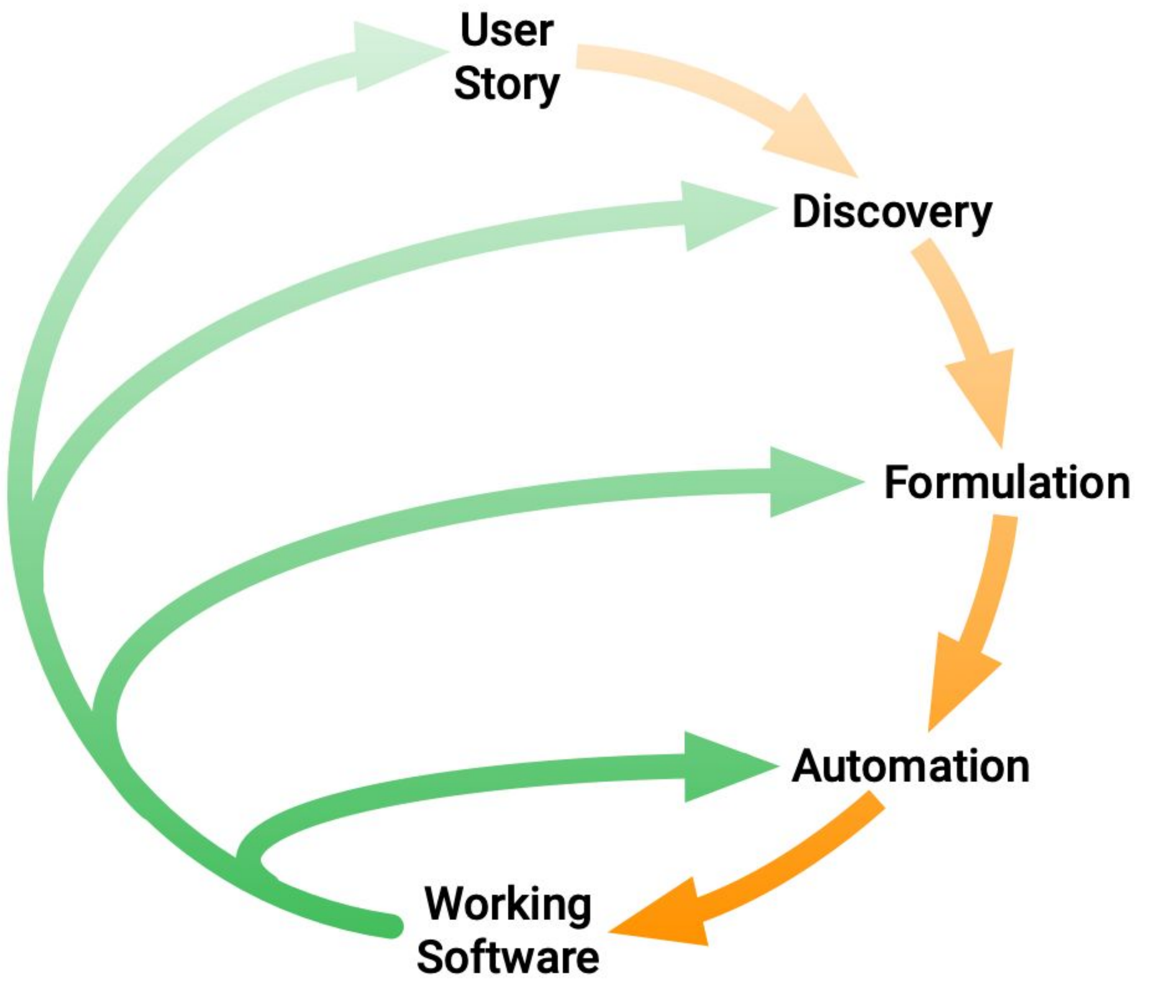
Initial Test Failure: Since the implementation is not yet complete, the tests will fail. This failure is expected at this stage.

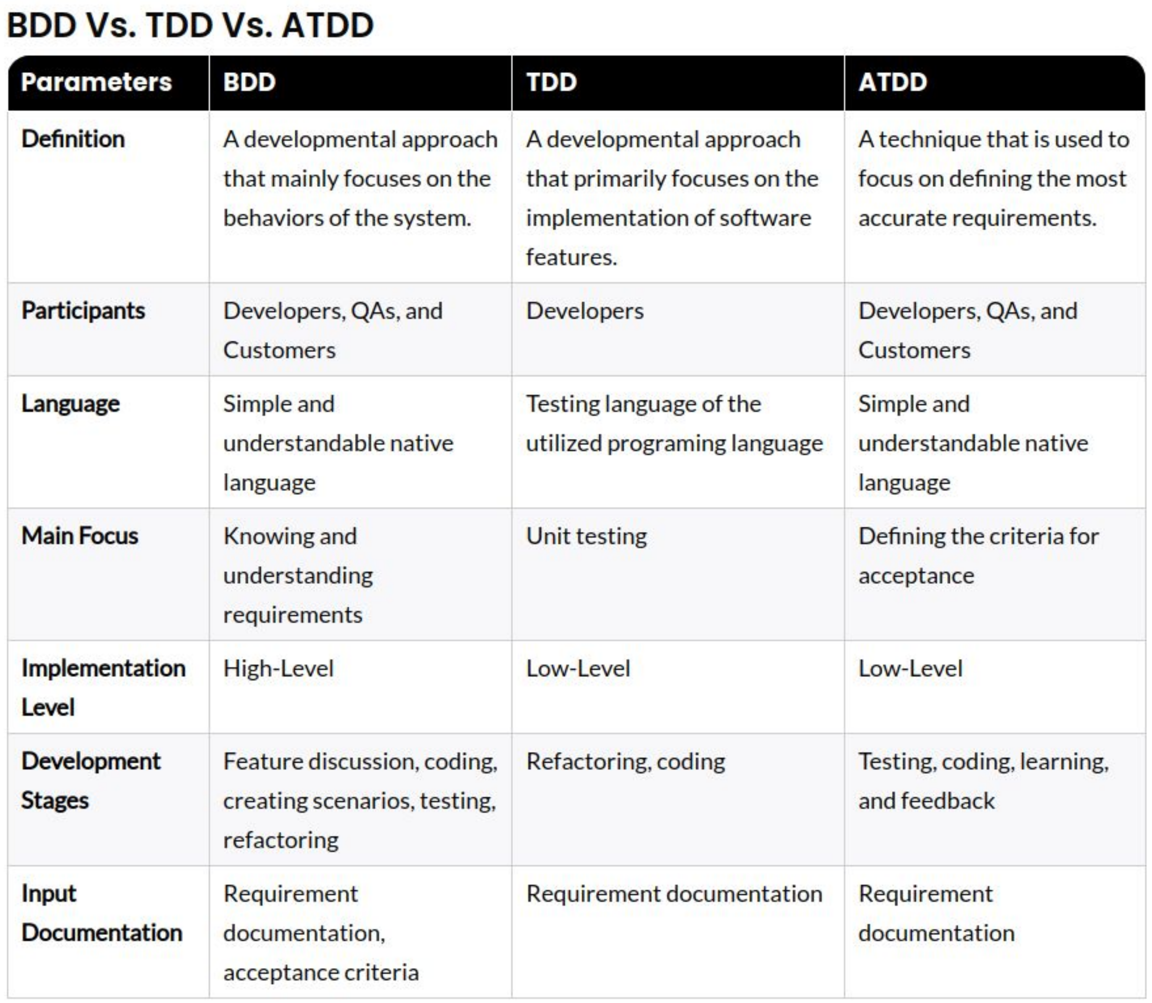
Implement the Feature: Start implementing the necessary code to make the tests pass. This typically involves developing the user registration functionality, including the registration form, backend validation, and email confirmation logic.

Refactor and Rerun the Tests: After implementing the feature, refactor the code as needed and rerun the BDD tests to ensure everything is functioning correctly.

Test Success: The tests should now pass, indicating that the user registration feature is implemented successfully.

Repeat the Process: Continue the BDD cycle by identifying new features or functionalities, creating feature files, implementing step definitions, and executing tests.


User Story
Discovery
Formulation
Automation
Working Software

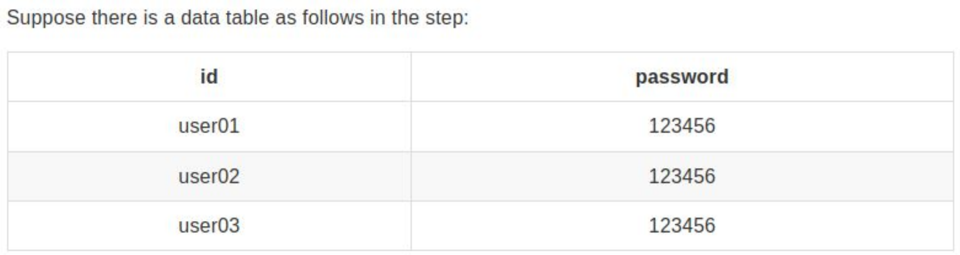


BDD Vs. TDD Vs. ATDD

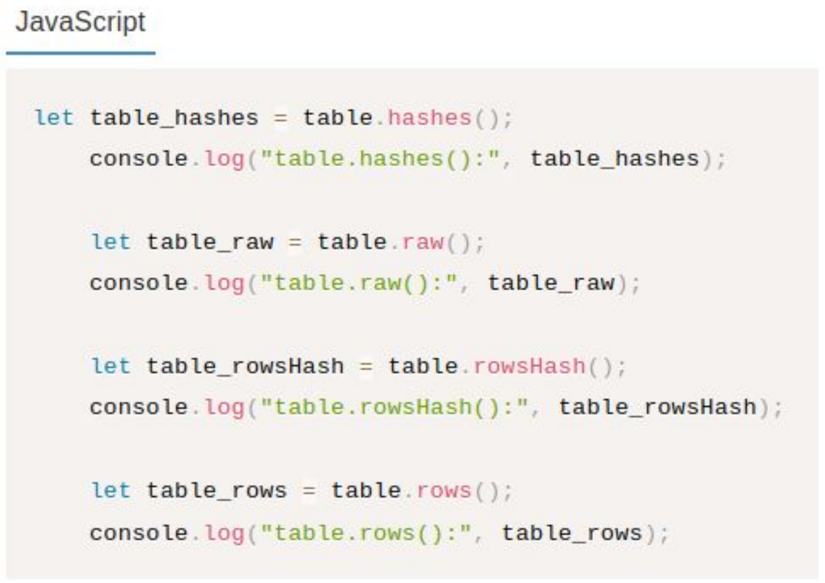
Parameters	BDD	TDD	ATDD
Definition	A developmental approach that mainly focuses on the behaviors of the system.	A developmental approach that primarily focuses on the implementation of software features.	A technique that is used to focus on defining the most accurate requirements.
Participants	Developers, QAs, and Customers	Developers	Developers, QAs, and Customers
Language	Simple and understandable native language	Testing language of the utilized programing language	Simple and understandable native language
Main Focus	Knowing and understanding requirements	Unit testing	Defining the criteria for acceptance
Implementation Level	High-Level	Low-Level	Low-Level
Development Stages	Feature discussion, coding, creating scenarios, testing, refactoring	Refactoring, coding	Testing, coding, learning, and feedback
Input Documentation	Requirement documentation, acceptance criteria	Requirement documentation	Requirement documentation

Part-2

Datatable in Steps:


Suppose there is a data table as follows in the step:

id	password
user01	123456
user02	123456
user03	123456

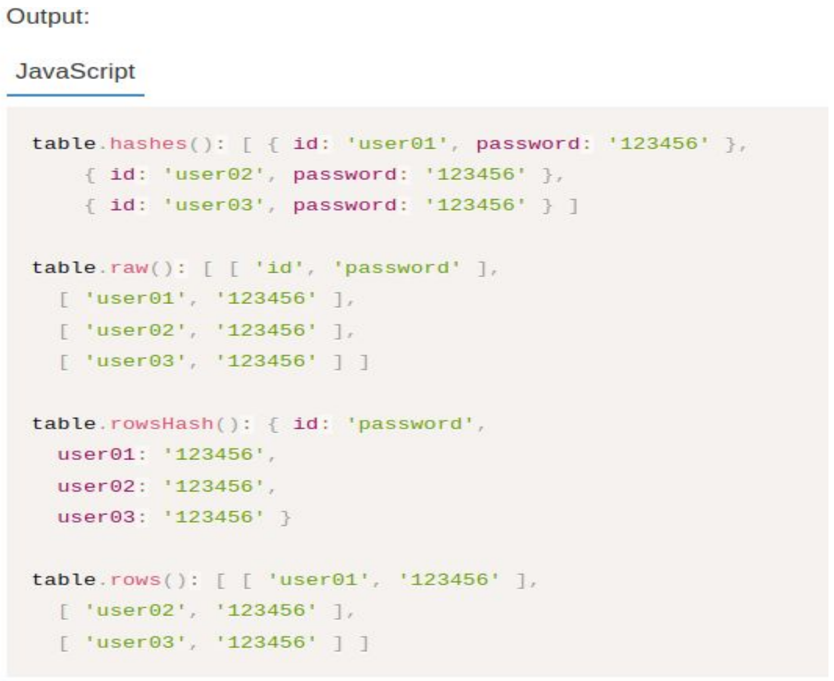

JavaScript

```
let table_hashes = table.hashes();
    console.log("table.hashes():", table_hashes);

    let table_raw = table.raw();
    console.log("table.raw():", table_raw);

    let table_rowsHash = table.rowsHash();
    console.log("table.rowsHash():", table_rowsHash);

    let table_rows = table.rows();
    console.log("table.rows():", table_rows);
```


Output:

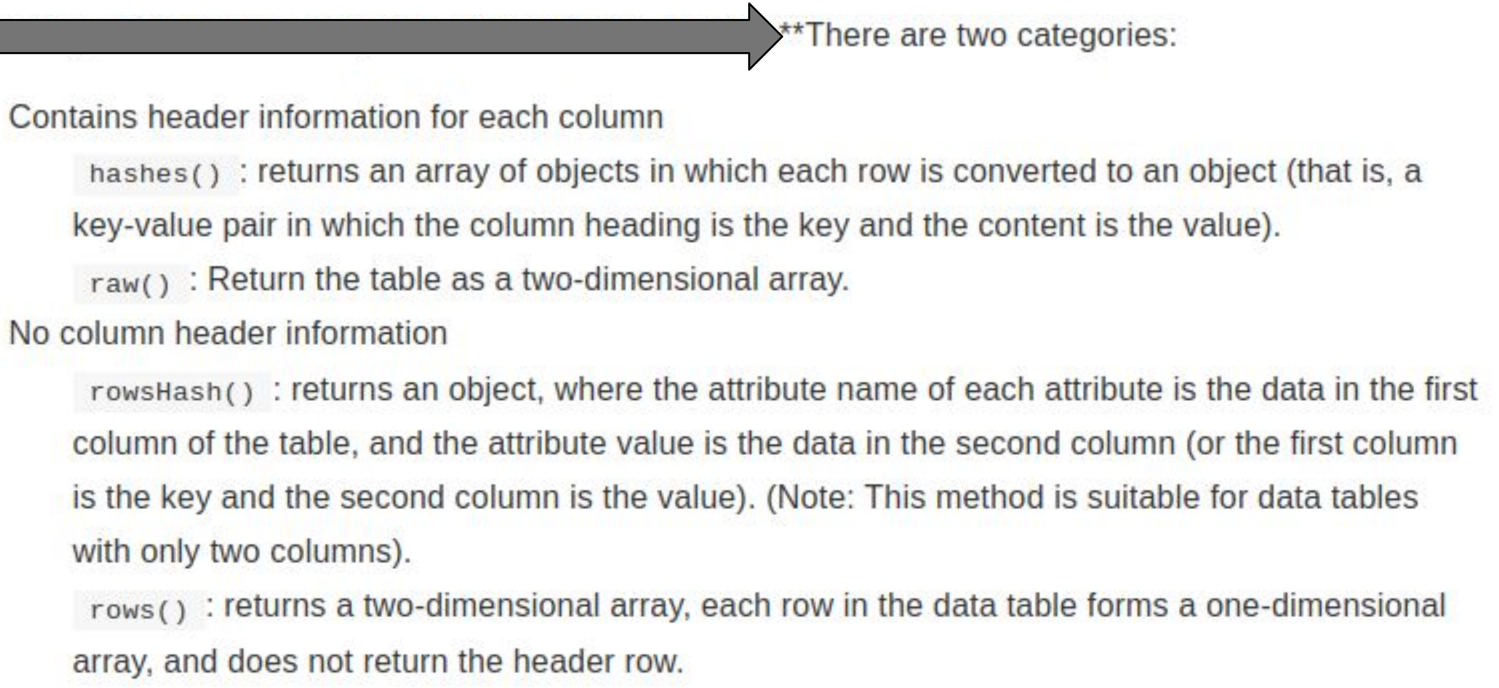
JavaScript

```
table.hashes(): [ { id: 'user01', password: '123456' },
    { id: 'user02', password: '123456' },
    { id: 'user03', password: '123456' } ]

table.raw(): [ [ 'id', 'password' ],
  [ 'user01', '123456' ],
  [ 'user02', '123456' ],
  [ 'user03', '123456' ] ]

table.rowsHash(): { id: 'password',
  user01: '123456',
  user02: '123456',
  user03: '123456' }

table.rows(): [ [ 'user01', '123456' ],
  [ 'user02', '123456' ],
  [ 'user03', '123456' ] ]
```



**There are two categories:

Contains header information for each column

`hashes()` : returns an array of objects in which each row is converted to an object (that is, a key-value pair in which the column heading is the key and the content is the value).

`raw()` : Return the table as a two-dimensional array.

No column header information

`rowsHash()` : returns an object, where the attribute name of each attribute is the data in the first column of the table, and the attribute value is the data in the second column (or the first column is the key and the second column is the value). (Note: This method is suitable for data tables with only two columns).

`rows()` : returns a two-dimensional array, each row in the data table forms a one-dimensional array, and does not return the header row.

Let’s Practice!

https://github.com/mnhmilu/poc-cucumberjs-bdd

References:

https://clustox.com/blog/what-is-bdd-the-role-scope-and-benefits-of-bdd-in-software-development/

https://cucumber.io/docs/bdd/who-does-what/

https://cucumber.io/docs/bdd/history/

https://cucumber.io/docs/guides/10-minute-tutorial/?lang=java

Thank you!