# Cucumber Cheat Sheet

A comprehensive guide to Cucumber, covering Gherkin syntax, step definitions, configuration, and best practices for writing effective and maintainable automated tests.

## Gherkin Syntax Essentials

### Feature and Scenario Structure

`Feature:` Describes a high-level feature of the application.

`Scenario:` A specific example of how the feature should behave.

`Scenario Outline:` A template for multiple scenarios with different data.

`Examples:` Table of data used with Scenario Outline.

**Example:**

```
Feature: User Authentication
  Scenario: Successful login
    Given User is on the login page
    When User enters valid credentials
    Then User should be logged in
```

### Keywords

| | |
|---|---|
| `Given` | Sets up the initial context of the scenario. |
| `When` | Describes an event or action performed by the user. |
| `Then` | Specifies the expected outcome or result. |
| `And`, `But` | Used to chain multiple `Given`, `When`, or `Then` steps for readability. |
| `Background` | A set of steps that run before each scenario in a feature. |

### Data Tables and Doc Strings

`Data Tables:` Used to pass structured data to a step definition.

`Doc Strings:` Used to pass larger blocks of text to a step definition.

**Data Table Example:**

```
Given the following users exist:
  | username | password |
  | john     | secret   |
  | jane     | password |
```

**Doc String Example:**

```
Given the following message:
  """
  This is a long message
  that spans multiple lines.
  """
```

## Step Definitions

### Basic Step Definition Structure

Step definitions link Gherkin steps to code that executes those steps.

```
Given('User is on the login page') do
  # Code to navigate to the login page
end
```

Step definitions typically use regular expressions to match the Gherkin step text.

### Regular Expression Usage

| | |
|---|---|
| `.*` | Matches any character (except newline) zero or more times. |
| `(\d+)` | Matches one or more digits and captures the value. |
| `([^"]*)` | Matches any character except a double quote, zero or more times, and captures the value. |
| ^(.*)$ | Matches the entire line and captures it. |

### Step Definition with Arguments

```
Given('User enters {string} as username') do |username|
  # Code to enter the username
  fill_in('username', with: username)
end
```

```
Given('the product name is {word}') do |product_name|
  # ...
end
```

## Configuration and Hooks

### Cucumber Configuration

Cucumber is typically configured using a `cucumber.yml` file or command-line options.

Key configuration options include:

- `paths`: Specifies the location of feature files.
- `requires`: Specifies files to load before running tests (e.g., step definitions, support files).
- `profiles`: Defines different configurations for different environments (e.g., test, development).

Example `cucumber.yml`:

```
default: --format pretty
test: --format progress --tags @test
```

### Hooks

| | |
|---|---|
| `Before` | Runs before each scenario or a tagged scenario. |
| `After` | Runs after each scenario or a tagged scenario. |
| `Around` | Wraps around each scenario, allowing you to perform actions before and after the scenario. |
| `AfterStep` | Runs after each step. |

### Hook Examples

```
Before('@database') do
  # Code to set up the database
end
```

```
After do |scenario|
  # Code to take a screenshot if the scenario fails
  if scenario.failed?
    save_screenshot('screenshot.png')
  end
end
```

## Advanced Cucumber Techniques

### Tagged Hooks and Scenarios

Tags are used to organize and filter scenarios and hooks.

Scenarios can be tagged directly in the feature file:

```
@smoke
Scenario: Successful login
  ...
```

Hooks can be tagged to run only for specific scenarios:

```
Before('@smoke') do
  # Code to run before smoke tests
end
```

### Parallel Execution

Cucumber can be configured to run scenarios in parallel, significantly reducing test execution time.

This often involves using a gem like `cucumber-parallel` or `parallel_tests`.

Configuration typically involves specifying the number of parallel processes to use.

### Best Practices

- **Write clear and concise Gherkin features:** Features should be easy to understand by both technical and non-technical stakeholders.
- **Keep step definitions focused:** Step definitions should perform a single, well-defined action.
- **Avoid duplication:** Use hooks and helper methods to avoid repeating code in step definitions.
- **Use data tables and doc strings effectively:** These features can help make your scenarios more readable and maintainable.
- **Run tests frequently:** Integrate Cucumber tests into your CI/CD pipeline to catch issues early.