bbv
MAKING VISIONS WORK.

Cucumber-cpp Cheat Sheet

Regular Expressions

Pattern	Notes	Match Examples

`.`	one of anything (except a newline)	`a`
		`A`
		`2`

`.*`	any character (except a newline) 0 or more times	`a`
		`ABcD`
		`words with punctuation!`
		`123-456`
		an empty string

`.+`	at least one of anything (except a newline)	all of the above except the empty string

`.{2}`	exactly two of any character	`ab`
		`AA`
		`??`
		`12`

`.{1,3}`	one to three of any character	`ab`
		`AA`
		`!n2`
		`1`

`^`*pattern*	anchors match to beginning of string	`/^foo/` matches `foo` and `foo bar` but not `bar foo`
pattern`$`	anchors match to end of string	`/foo$/` matches `foo` and `bar foo` but not `foo bar`

`[0-9]*` or `\d*`	matches a series of digits (or nothing)	`123456`
		`9`
		an empty string

`[0-9]+` or `\d+`	matches a series of digits (or nothing)	all of the above except the empty string

`"[^\"]*"`	matches something (or nothing) in double quotes; literally, “any character except a double quote zero or more times inside double quotes”	`"foo"`
		`"foo bar"`
		`"123456"`

`?`	makes the previous character or group optional	`/an?/` matches `a` and `an` but not `n`
`|`	like a logical OR; can be used with parentheses to include an OR in a larger pattern	`/I'm|I am/` matches `I'm` and `I am` `/I (eat|ate)/` matches `I eat` and `I ate`
(*pattern*)	a group; typically used to capture a substring for a step definition argument	`/(\d+) users/` matches `3 users` and captures the `3` for later use
(?:*pattern*)	a non-capturing group	`/I (?:eat|ate)/` matches `I eat` and `I ate` but does not capture `eat` or `ate` for later use

Cucumber-cpp regex example

```
GIVEN("^light (\\d+) is scheduled to turn \"(ON|OFF)\" at (\\d{2,2}):(\\d{2,2})$"){ /*…*/ }
```

Cucumber-cpp Example

```
# features/calculator.feature
# language: en

Feature: Cucumber-cpp Academy Exercise
  As a C++ developer with first time contact with Cucumber-cpp
  I want to implement all kind of cucumber steps without interaction to other code
  In order to learn how to define cucumber steps in C++

Scenario: Simple Steps
  Given the numbers 6 and 7
  When this numbers are multiplied
  Then the result should be 42

Scenario Outline: Examples
  Given the numbers <number 1> and <number 2>
  When this numbers are added
  Then the result should be <result>

Examples:
  | number 1  | number 2  | result  |
  |     4     |     25    |    29   |
  |     0     |      0    |     0   |
  |   -20     |     25    |     5   |

Scenario: Input Table
  Given the following numbers:
    | number |
    |    23  |
    |    34  |
    |   100  |

  When each number is multiplied by 2
  Then the results should be the following:
    | result |
    |    46  |
    |    68  |
    |   200  |

Scenario: String Input
  Given the following text
    """
    Then the Queen left off, quite out of breath, and said to Alice,
    'Have you seen the Mock Turtle yet?'
    'No,' said Alice. 'I don't even know what a Mock Turtle is.'
    'It's the thing Mock Turtle Soup is made from,' said the Queen.
    'I never saw one, or heard of one,' said Alice.
    'Come on, then,' said the Queen, 'and he shall tell you his history,'
    """
  When the word Mock are counted in the text
  Then the count should be 3
```

```cpp
// features/step_definitions/calculator_steps.cpp
#include <gtest/gtest.h>
#include <cucumber-cpp/autodetect.hpp>
#include <string>
#include <vector>
#include <algorithm>

namespace {

using cucumber::ScenarioScope;

struct CalculatorCtx {
    int first_number{};
    int second_number{};
    int result{};
};

GIVEN("^the numbers (-?\\d+) and (-?\\d+)$") {
    REGEX_PARAM(int, first_number);
    REGEX_PARAM(int, second_number);

    ScenarioScope<CalculatorCtx> context{};
    context->first_number = first_number;
    context->second_number = second_number;
}

struct CalculatorCtx2 {
    std::vector<int> numbers{};
};

GIVEN("^the following numbers:$") {
    ScenarioScope<CalculatorCtx2> context{};

    TABLE_PARAM(number_params);
    const auto& number_table = number_params.hashes();

    for (const auto& table_row : number_table) {
        context->numbers.push_back(std::stoi(table_row.at("number")));
    }
}

struct TextCtx {
    std::string input_text{};
    unsigned int counter{};
};

GIVEN("^the following text$") {
    REGEX_PARAM(std::string, text); // text is the last parameter passed

    ScenarioScope<TextCtx> context{};
    context->input_text = text;
}

WHEN("^this numbers are multiplied$") {
    ScenarioScope<CalculatorCtx> context{};
    context->result = context->first_number * context->second_number;
}
```

```
WHEN("^this numbers are added$") {
    ScenarioScope<CalculatorCtx> context{};
    context->result = context->first_number + context->second_number;
}

WHEN("^each number is multiplied by (-?\\d)$") {
    REGEX_PARAM(int, multiplier);
    ScenarioScope<CalculatorCtx2> context{};
    std::transform(std::begin(context->numbers),
                   std::end(context->numbers),
                   std::begin(context->numbers),
                   [&multiplier](int value) { return value * multiplier;});
}

WHEN("^the word (.+) are counted in the text$") {
    pending();
}

THEN("^the result should be (-?\\d+)$") {
    REGEX_PARAM(int, result);
    ScenarioScope<CalculatorCtx> context;
    ASSERT_EQ(result, context->result);
}

THEN("^the results should be the following:$") {
    TABLE_PARAM(number_params);

    ScenarioScope<CalculatorCtx2> context{};

    const auto& number_table = number_params.hashes();

    std::vector<int> results;

    for (const auto& table_row : number_table)
    {
       results.push_back(std::stoi(table_row.at("result")));
    }
    ASSERT_TRUE(std::equal(std::begin(results), std::end(results),
std::begin(context->numbers)));
}

THEN("^the count should be (\\d)$") {
    pending();
}

}//namespace
```

Tips & Tricks

Hooks

```
BEFORE_ALL(){ /* Before all scenarios */ }
AFTER_ALL (){ /* After all scenarios */ }

BEFORE(){ /* Before any scenarios */ }
AFTER(){ /* After any scenarios */ }

BEFORE("@light_control"){ /* Before scenarios tagged @light_control */ }
AFTER("@light_control"){ /* After scenarios tagged @light_control */ }

BEFORE("@foo,@bar"){ /* Before scenarios tagged @foo AND @bar */ }

AFTER("@foo","@bar"){ /* After scenarios tagged @foo OR @bar */ }
```

Background

```
Background:
  Given the light 0 is "OFF“
  And the light 1 is "ON"

Scenario: Turn a single light On
  When I switch the light 0 "ON"
  Then the light 0 should be "ON"

Scenario: Turn a single light Off
  When I switch the light 1 "OFF"
  Then the light 1 should be "OFF"
```

Work in Progress

Run only features NOT tagged @wip:

```
  #>cucumber --tags ~@wip features
```

Run only features tagged @wip, limit WIP to 3 and fail if scenarios pass.

```
  #>cucumber --tags @wip:3 --wip features
```

Running a subset of features

A single feature:

```
  #>cucumber features/light_control/light_switch.feature
```

A single scenario:

```
  #>cucumber features/light_control/light_switch.feature:42
```

A group of features:

```
  #>cucumber features/light_control
```

Run only features tagged @regression OR @fast AND @light_controller:

```
  #>cucumber --tags @regression,@fast --tags @light_controller features
```