RSpec::Expectations Cheat Sheet Ruby

Basic Matchers

```
RSpec.describe 'Common, built-in expectation matchers' do
  example 'Equality' do
    expect('x'+'y').to     eq('xy')   # a == b
    expect('x'+'y').to     eql('xy')  # a.eql?(b)
    expect('x'+'y').not_to be('xy')   # a.equal?(b)
  end

  example 'Strings' do
    expect('abcd').to include('bc')
    expect('abcd').to start_with 'ab'
    expect('abcd').to end_with 'cd'
    expect('abcd').to match /[a-z]+/
  end

  example 'Collections' do
    expect([1, 2, 3]).to include(1, 3)
    expect([1, 2, 3]).to contain_exactly(3, 2, 1) # order not important
    expect({ a: 1, b: 2 }).to include(b: 2)
  end

  example 'Booleans and nil' do
    expect(true).to be true
    expect(false).to be false
    expect('abc').to be_truthy
    expect(nil).to be_falsey
    expect(nil).to be_nil
  end

  example 'Numeric' do
    expect(5).to be > 4
    expect(5).to be >= 4
    expect(5).to be < 6
    expect(5).to be <= 6
    expect(5).to be_between(4, 6).exclusive
    expect(5).to be_between(5, 6).inclusive
    expect(4.99).to be_within(0.02).of(5)
  end

  example 'Errors (exceptions)' do
    expect{ 5 / 0 }.to raise_error(ZeroDivisionError)
    expect{ 5 / 0 }.to raise_error("divided by 0")
    expect{ 5 / 0 }.to raise_error(ZeroDivisionError, "divided by 0")
```

```
  end
end
```

Predicate Matchers

Predicate matchers are a little DSL for calling predicate methods. Predicate methods are methods that:

1. return a boolean value; and
2. have a name that ends with `?`

Commonly used predicate methods in the Ruby standard library include: `Object#nil?`, `Array#empty?`, and `Hash#has_key?`.

```
RSpec.describe 'Predicate matchers' do
  example 'Array' do
    expect([]).to be_empty          # [].empty?
  end

  example 'Hash' do
    expect({a: 1}).to have_key(:a)   # {a: 1}.has_key?(:a)
    expect({a: 1}).to have_value(1)  # {a: 1}.has_value?(1)
  end

  example 'Object' do
    expect(5).not_to be_nil             # 'hi'.nil?
    expect(5).to be_instance_of Fixnum  # 5.instance_of?(Fixnum)
    expect(5).to be_kind_of Numeric     # 5.kind_of?(Numeric)
  end
end
```

Predicate matchers work on all objects, including custom classes.

```
classWidget
attr_accessor :name, :cost

def initialize(name, cost)
    @name= name
    @cost= cost
end
```

```
def has_cliche_name?
  ['Foo', 'Bar', 'Baz'].include?(@name)
end

def hacker?
  @cost== 1337
  end
end

RSpec.describe 'Predicate matchers'do
  example 'With a custom class'do
    widget= Widget.new('Foo', 1337)

    expect(widget).to have_cliche_name
    expect(widget).to be_hacker
    expect(widget).to be_a_hacker
    expect(widget).to be_an_hacker
 end
end
```

Advanced Matchers

These are the more complicated matchers, that aren’t as commonly used

```
# Add one extra method to the Widget class above,
# for demonstrating change observation.

class Widget
  def fifty_percent_off!
   @cost/= 2
  end
end

RSpec.describe 'Advanced matchers'do
  example 'Change observation'do
   widget= Widget.new('Baz', 80)

    expect{ widget.has_cliche_name? }.not_to change(widget, :name)

    expect{ widget.fifty_percent_off! }.to change(widget, :cost)
    # 80 -> 40expect{ widget.fifty_percent_off! }.to change(widget, :cost).from(40).to
(20)

    expect{ widget.fifty_percent_off! }.to change(widget, :cost).by(-10)
    # 20 -> 10endexample 'Object attributes'do# The attributes are a hash of method na
mes to
    # expected return values.expect('hi').to have_attributes(length: 2, upcase: 'HI')
   end
```

```
example 'Yielding to blocks'do
    expect{|b| 5.tap(&b) }.to yield_control
    expect{|b| 5.tap(&b) }.to yield_with_args(5)
    expect{|b| 5.tap(&b) }.to yield_with_args(Integer)
    expect{|b| 5.tap(&b) }.not_to yield_with_no_args
    expect{|b| 3.times(&b) }.to yield_successive_args(0, 1, 2)
end

   example 'Output capture'do
     expect{ puts 'hi' }.to output("hi\n").to_stdout
     expect{ $stderr.puts 'hi' }.to output("hi\n").to_stderr
   end

   example 'Throws'do
     # Not to be confused with errors (exceptions).
     # Throw/catch is very rarely used.expect{throw :foo, 5 }.to throw_symbol
     expect{throw :foo, 5 }.to throw_symbol(:foo)
     expect{throw :foo, 5 }.to throw_symbol(:foo, 5)
   end

  example 'All elements in a collection'do
    expect([3,5,7]).to all(be_odd)
    expect([3,5,7]).to all(be> 0)
   end

  example 'Compound matchers'do
    expect(5).to be_odd.and be> 0
    expect(5).to be_odd.or be_even

    # These are probably most useful in combination with
    # the `all` matcher (above). For example:expect([1,2,3]).to all(be_kind_of(Intege
r).and be> 0)
  end

  example 'Satisfy'do
    # Test against a custom predicateexpect(9).to satisfy('be a multiple of 3'){|x| x%
3== 0 }
  end
end
```