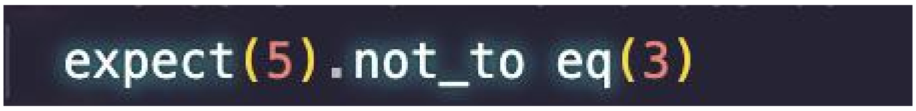
33. The not_to Method

There are multiple matchers (one of them is “eq”).
The inverse of “to” method

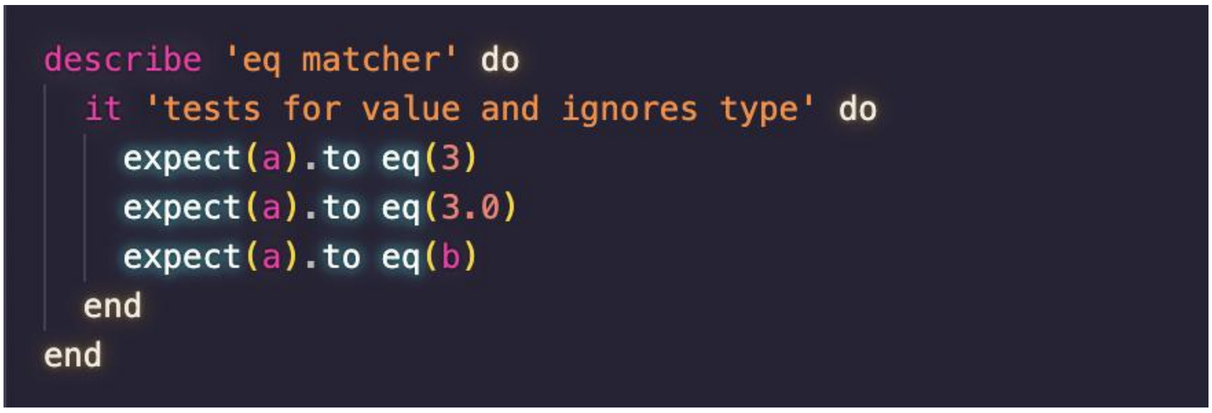


```ruby
expect(5).not_to eq(3)
```

34. Equality Matchers I (eq and eql)

eq matcher

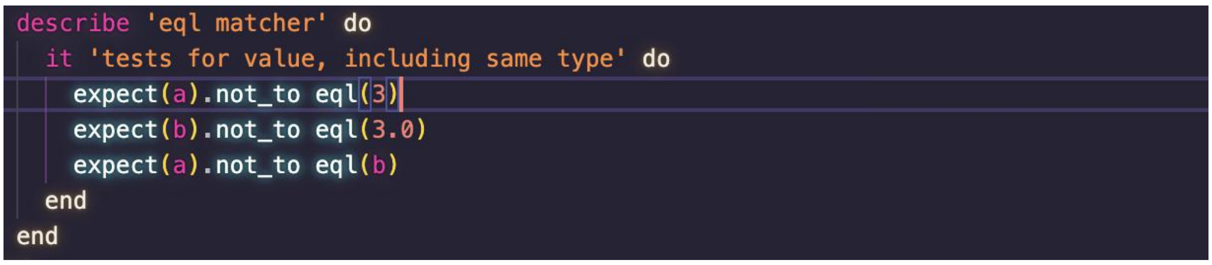
Checks the value and ignores type



```ruby
describe 'eq matcher' do
  it 'tests for value and ignores type' do
    expect(a).to eq(3)
    expect(a).to eq(3.0)
    expect(a).to eq(b)
  end
end
```

eql matcher

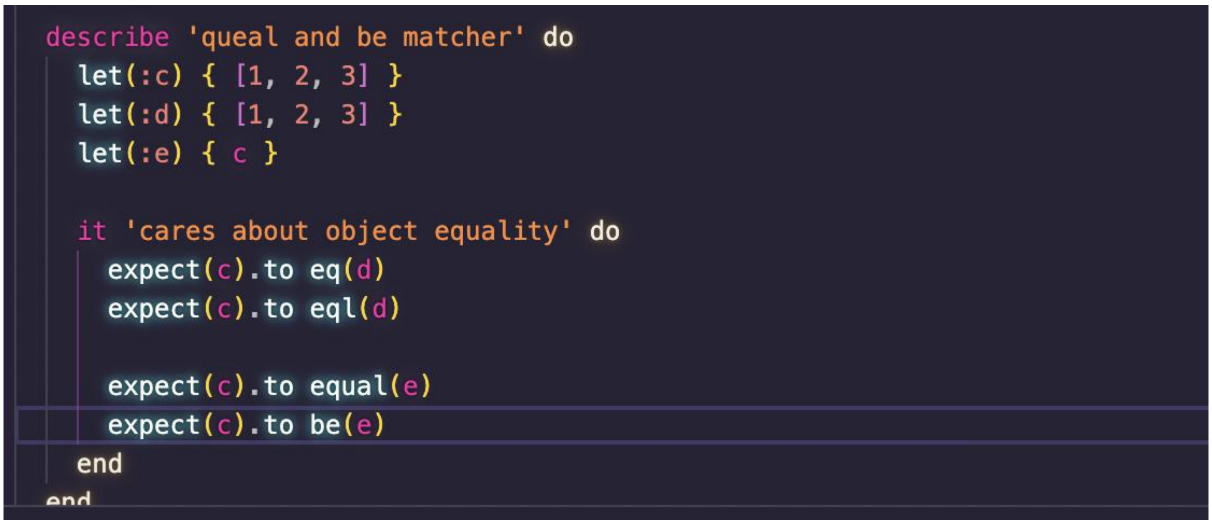
Checks value and checks the values to be the same type



```ruby
describe 'eql matcher' do
  it 'tests for value, including same type' do
    expect(a).not_to eql(3)
    expect(b).not_to eql(3.0)
    expect(a).not_to eql(b)
  end
end
```

35. Equality Matchers II (equal and be)

Quality is more about the value. Identity about if they are the same object.
be and equal are aliases.

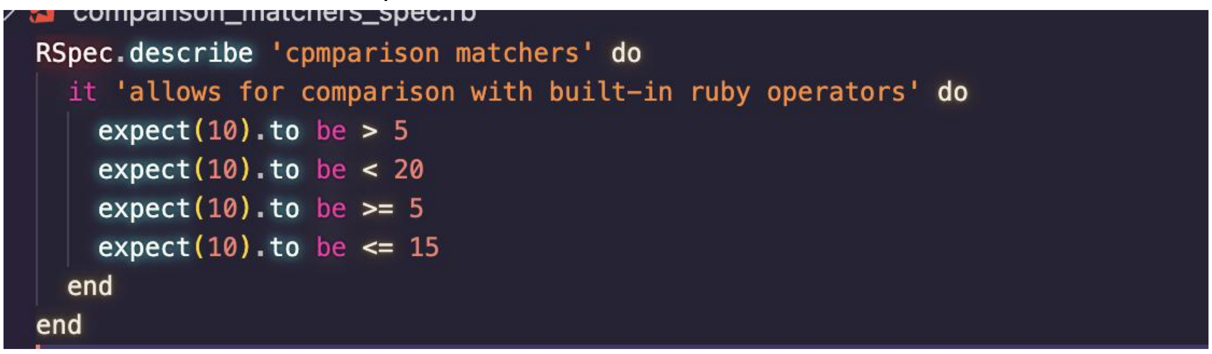


```ruby
describe 'queal and be matcher' do
  let(:c) { [1, 2, 3] }
  let(:d) { [1, 2, 3] }
  let(:e) { c }

  it 'cares about object equality' do
    expect(c).to eq(d)
    expect(c).to eql(d)

    expect(c).to equal(e)
    expect(c).to be(e)
  end
end
```

36. Comparison Matchers

Use ruby operators

Use method “be” before the operators

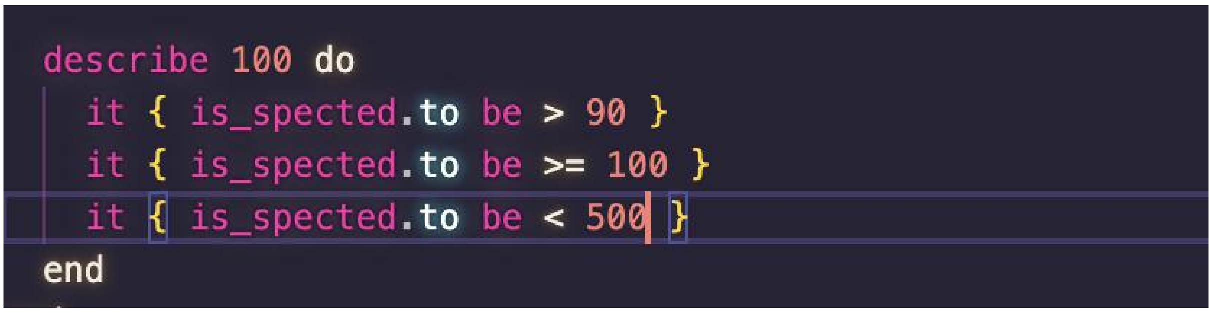


comparison_matchers_spec.rb

```ruby
RSpec.describe 'cpmparison matchers' do
  it 'allows for comparison with built-in ruby operators' do
    expect(10).to be > 5
    expect(10).to be < 20
    expect(10).to be >= 5
    expect(10).to be <= 15
  end
end
```

One line and use object instead of class

We can pass a specific object as well as the describe method parameter

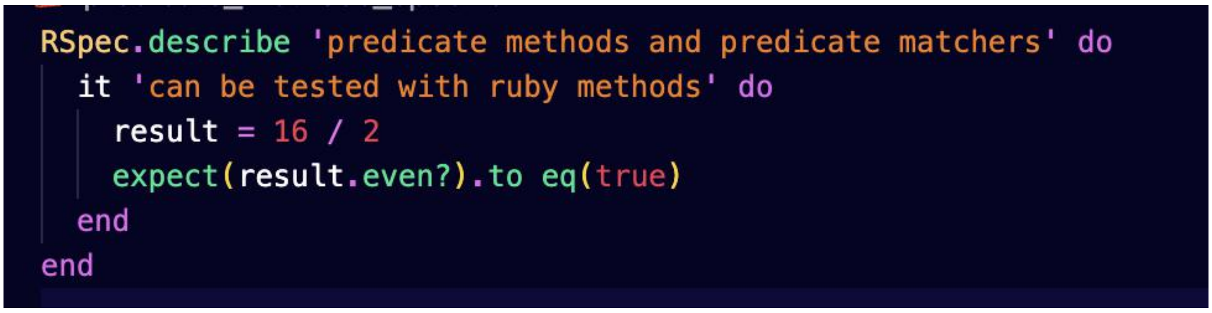


```ruby
describe 100 do
  it { is_spected.to be > 90 }
  it { is_spected.to be >= 100 }
  it { is_spected.to be < 500 }
end
```

37. Predicate Matchers

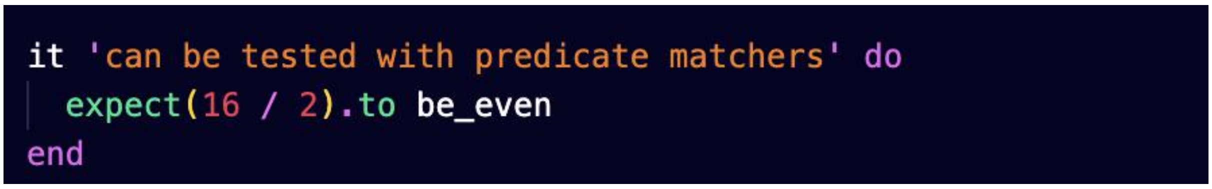
Predicate methods are the ones that return false or true. They end with a question mark (it is not a technical standard).

To include them on a test, you will do something like this:



```ruby
RSpec.describe 'predicate methods and predicate matchers' do
  it 'can be tested with ruby methods' do
    result = 16 / 2
    expect(result.even?).to eq(true)
  end
end
```

But we have a shortcut to use them in RSpec



```ruby
it 'can be tested with predicate matchers' do
  expect(16 / 2).to be_even
end
```

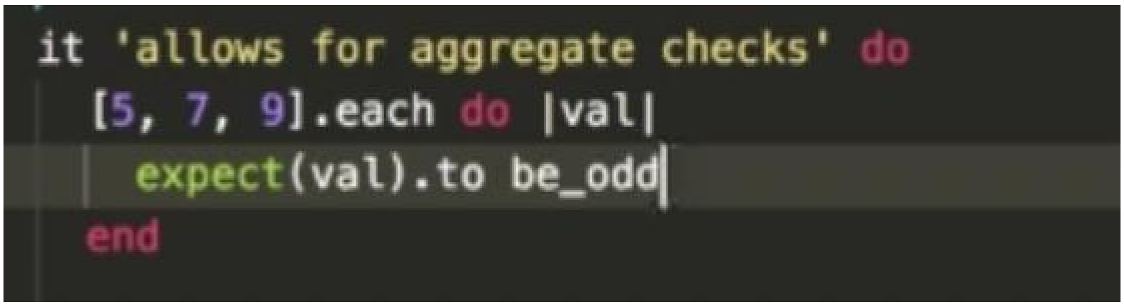
After “to” you add a predicate method from ruby, remove the interrogation mark and add the prefix “be_” to it.

For example, we have the predicate method “.even?”, we will write it down like “be_even”

- be_even
- be_odd
- be_zero
- be_empty
- etc

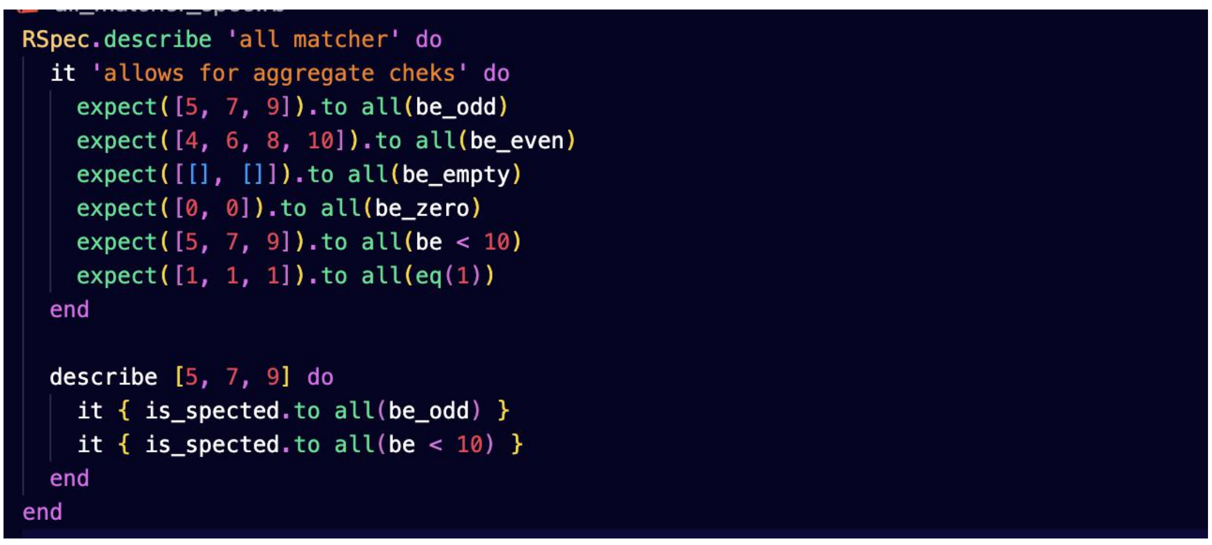
38. all Matcher

It is useful to check arrays.
<u>You will usually do this:</u>
Iterate over the array and assert each value



```ruby
it 'allows for aggregate checks' do
  [5, 7, 9].each do |val|
    expect(val).to be_odd
  end
```

<u>Better way:</u>
You use “all” and you pass it a matcher, a predicate method to be more specific.

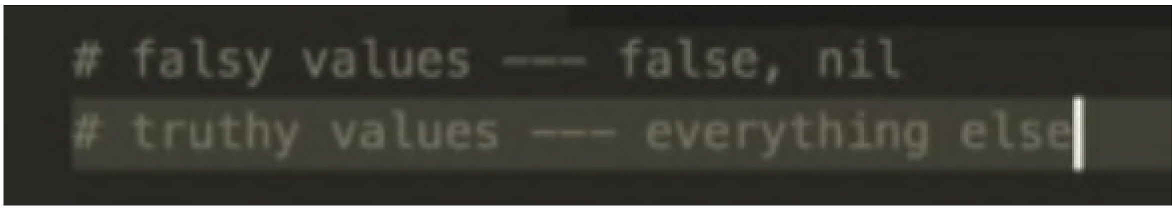


```ruby
RSpec.describe 'all matcher' do
  it 'allows for aggregate cheks' do
    expect([5, 7, 9]).to all(be_odd)
    expect([4, 6, 8, 10]).to all(be_even)
    expect([[], []]).to all(be_empty)
    expect([0, 0]).to all(be_zero)
    expect([5, 7, 9]).to all(be < 10)
    expect([1, 1, 1]).to all(eq(1))
  end

  describe [5, 7, 9] do
    it { is_spected.to all(be_odd) }
    it { is_spected.to all(be < 10) }
  end
end
```

It checks that all values of the array fulfill a condition.

39. be Matcher (Truthy, Falsy and Nil Values)

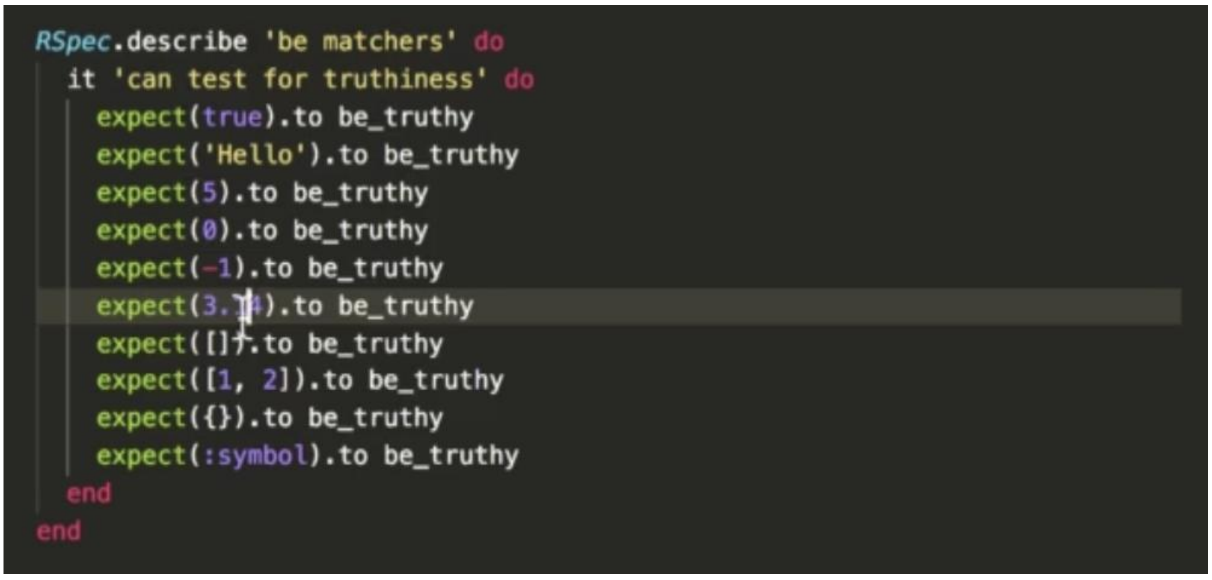
Truthy and falsy means passing non boolean values to a condition structure to check them.



```ruby
# falsy values --- false, nil
# truthy values --- everything else
```

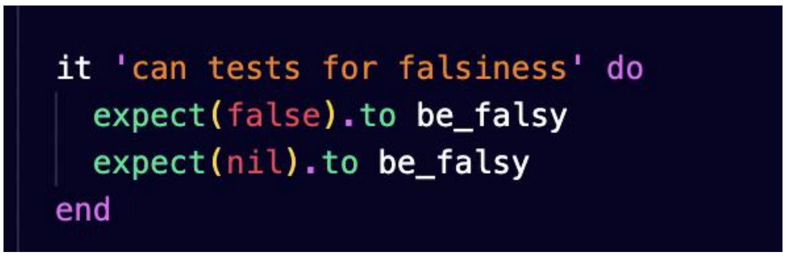
It can be used when you don't care what is being returned but you care about if it is valid.

be truthy



```ruby
RSpec.describe 'be matchers' do
  it 'can test for truthiness' do
    expect(true).to be_truthy
    expect('Hello').to be_truthy
    expect(5).to be_truthy
    expect(0).to be_truthy
    expect(-1).to be_truthy
    expect(3.14).to be_truthy
    expect([]).to be_truthy
    expect([1, 2]).to be_truthy
    expect({}).to be_truthy
    expect(:symbol).to be_truthy
  end
end
```

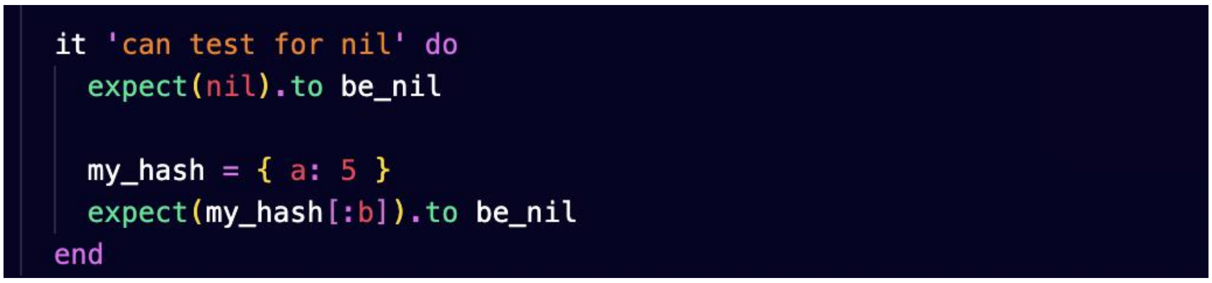
be falsy



```ruby
it 'can tests for falsiness' do
  expect(false).to be_falsy
  expect(nil).to be_falsy
end
```

be_nil

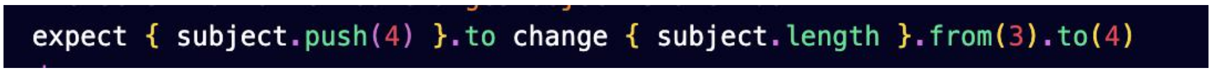
We can check if the value is nil on specific
Useful to check if a hash key exists



```ruby
it 'can test for nil' do
  expect(nil).to be_nil

  my_hash = { a: 5 }
  expect(my_hash[:b]).to be_nil
end
```

40. change Matcher

An operator that allows tracking change after a method call.



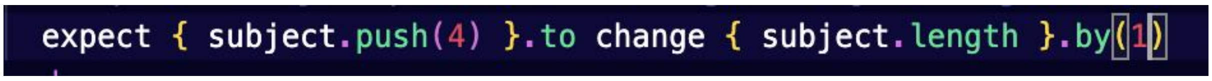
```ruby
expect { subject.push(4) }.to change { subject.length }.from(3).to(4)
```

On a block we pass the method that supposedly is going to make a change to an attribute. Then using change we pass a block where we add what value should change. Then using "from" and "to" we specify from what value and to which value it should change.

He does not like the tree and four hardcoded since if someone change the original size of the array it is not going to work

By

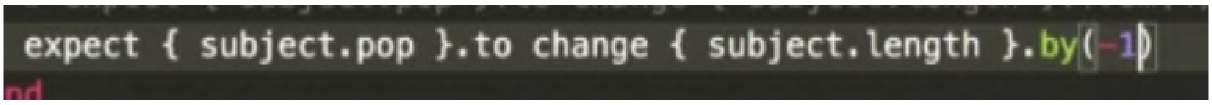
"by" will fix this issue, will read the initial value automatically and then we pass how much it will change



```ruby
expect { subject.push(4) }.to change { subject.length }.by(1)
```

A common use by this "change matcher" is when you add something to your database and you want to see if the quantity of rows increased

Also works with negatives (to check a decrease)



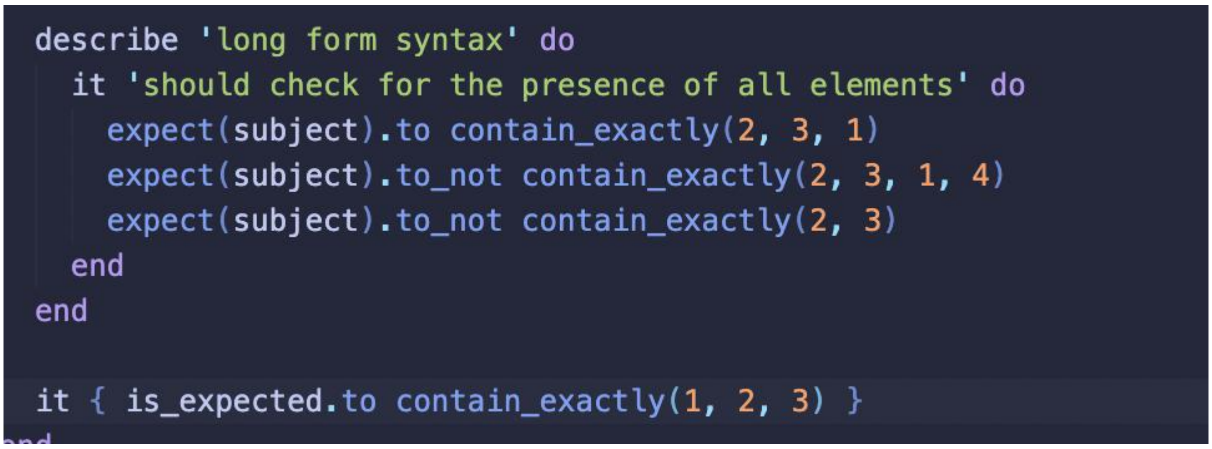
```
expect { subject.pop }.to change { subject.length }.by(-1)
```

Also works with 0 (to check it did not change)

41. contain_exactly Matcher

Checks if an array contains some elements in any order

It fails also if there are more elements than in the array, or less elements in the array

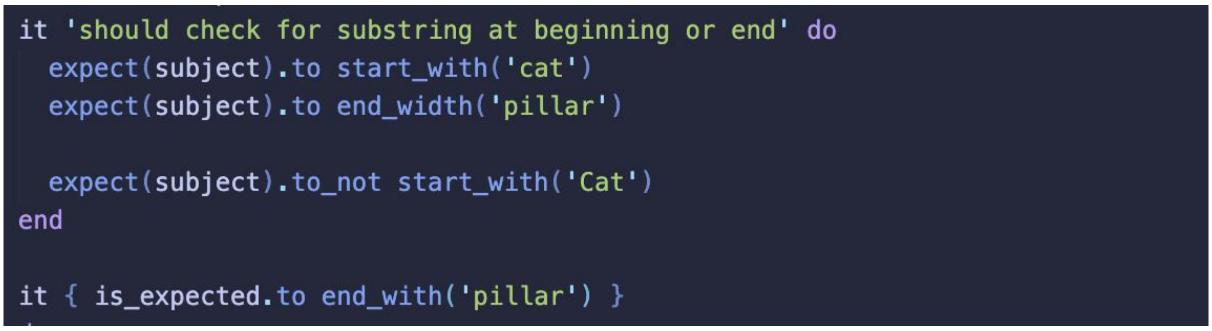


```
describe 'long form syntax' do
  it 'should check for the presence of all elements' do
    expect(subject).to contain_exactly(2, 3, 1)
    expect(subject).to_not contain_exactly(2, 3, 1, 4)
    expect(subject).to_not contain_exactly(2, 3)
  end
end

it { is_expected.to contain_exactly(1, 2, 3) }
```

If you want to check for the values and care about the order you can use the eq. But if you do not care then use this one.

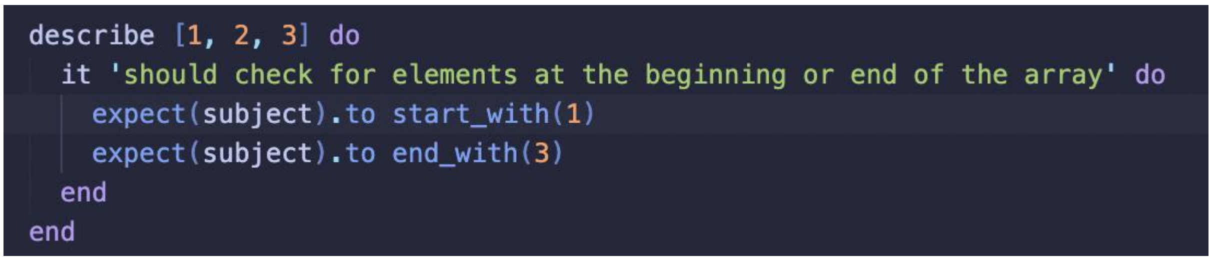
42. start_with and end_with Matchers

Checks if an object (string or array) starts or ends with a value.
It is case sensitive



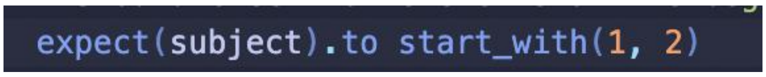
```
it 'should check for substring at beginning or end' do
  expect(subject).to start_with('cat')
  expect(subject).to end_width('pillar')

  expect(subject).to_not start_with('Cat')
end

it { is_expected.to end_with('pillar') }
```

Array:



```ruby
describe [1, 2, 3] do
  it 'should check for elements at the beginning or end of the array' do
    expect(subject).to start_with(1)
    expect(subject).to end_with(3)
  end
end
```

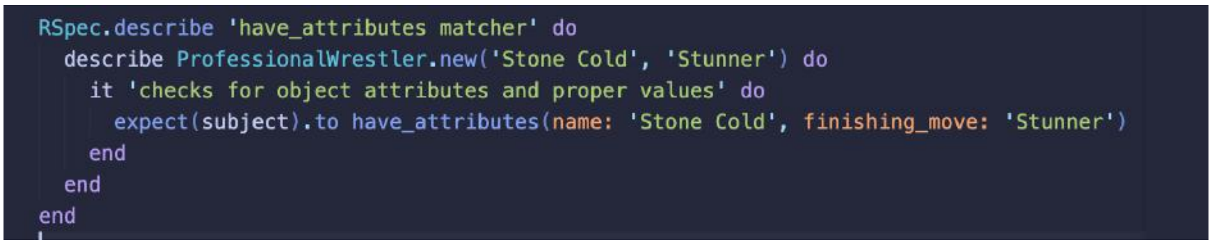
With an array, you can check for more than one value



```ruby
expect(subject).to start_with(1, 2)
```

43. have_attributes Matcher

It checks for attributes and their corresponding value



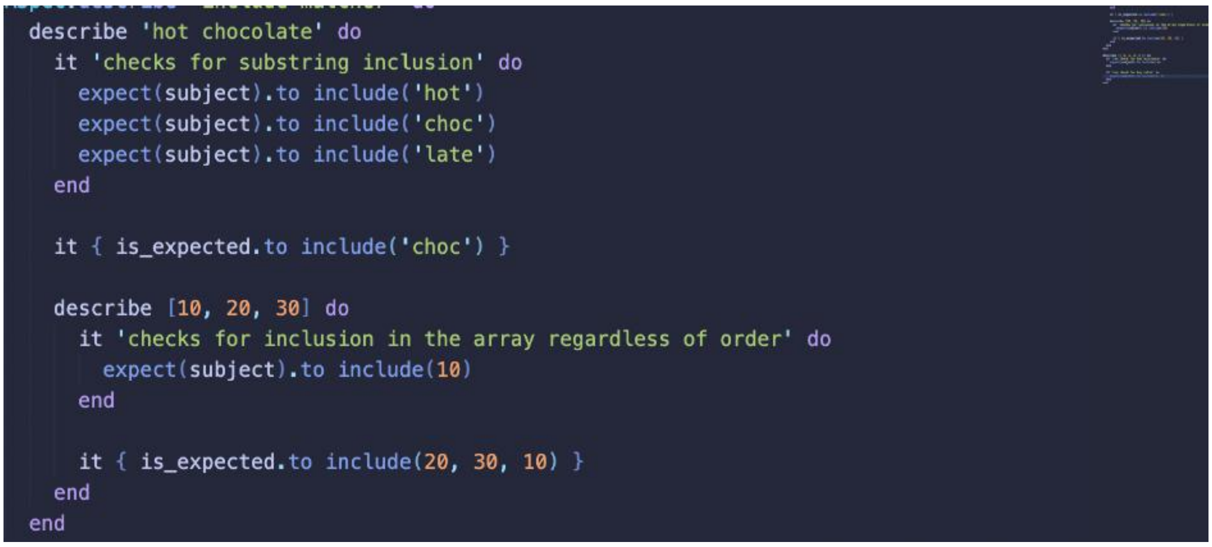
```ruby
RSpec.describe 'have_attributes matcher' do
  describe ProfessionalWrestler.new('Stone Cold', 'Stunner') do
    it 'checks for object attributes and proper values' do
      expect(subject).to have_attributes(name: 'Stone Cold', finishing_move: 'Stunner')
    end
  end
end
```

Ruby feature
If a hash represents the last argument from a method, then you can pass a hash without the curly brackets.

44. include Matcher

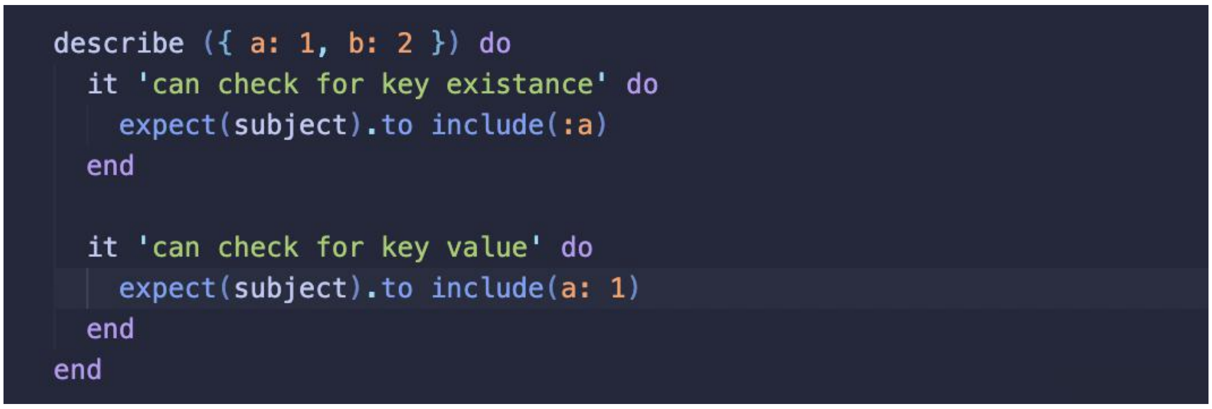
Check if array or string includes an element or substring
You can pass multiple parameters.

String and array



```ruby
  describe 'hot chocolate' do
    it 'checks for substring inclusion' do
      expect(subject).to include('hot')
      expect(subject).to include('choc')
      expect(subject).to include('late')
    end

    it { is_expected.to include('choc') }

    describe [10, 20, 30] do
      it 'checks for inclusion in the array regardless of order' do
        expect(subject).to include(10)
      end

      it { is_expected.to include(20, 30, 10) }
    end
  end
```

Hash



```ruby
describe ({ a: 1, b: 2 }) do
  it 'can check for key existance' do
    expect(subject).to include(:a)
  end

  it 'can check for key value' do
    expect(subject).to include(a: 1)
  end
end
```

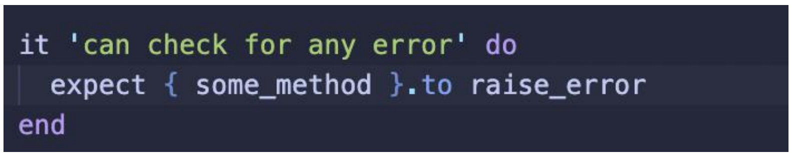
45. raise_error Matcher

Checks if something throws an error

We need to pass a block since if we call something that throws errors then it is going to fail, we pass the block to the expected method and it handles the error.

Test for any error

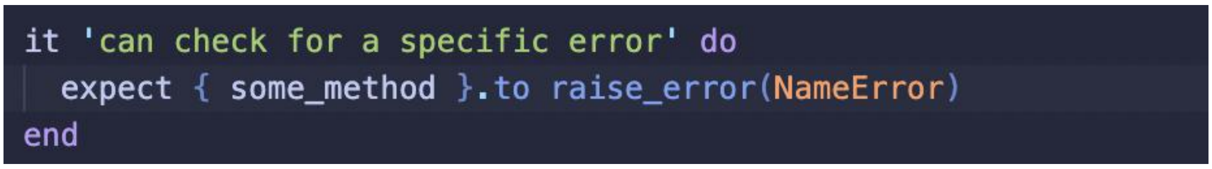
This is not recommended since it can throw an error that you are not expecting to be thrown and have a false positive.



```ruby
it 'can check for any error' do
  expect { some_method }.to raise_error
end
```

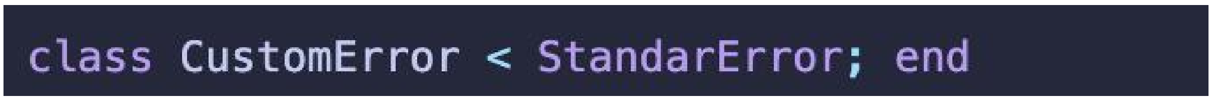
Test for a specific error

It can also check for custom errors.



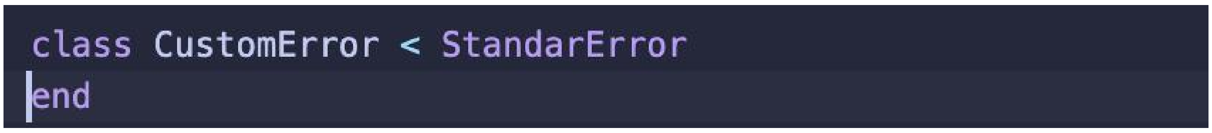
```
it 'can check for a specific error' do
  expect { some_method }.to raise_error(NameError)
end
```

Off topic, but here it is how to create your own exceptions



```
class CustomError < StandarError; end
```

or

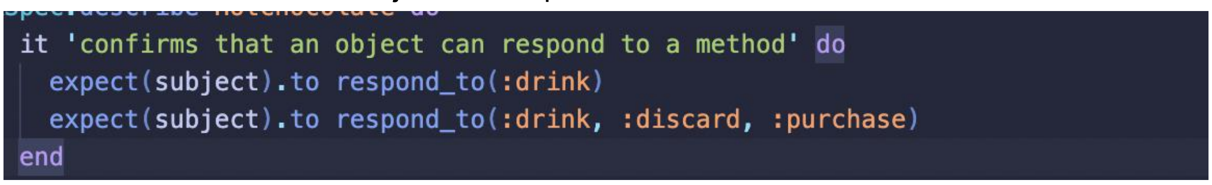


```
class CustomError < StandarError
end
```

46. respond_to Matcher

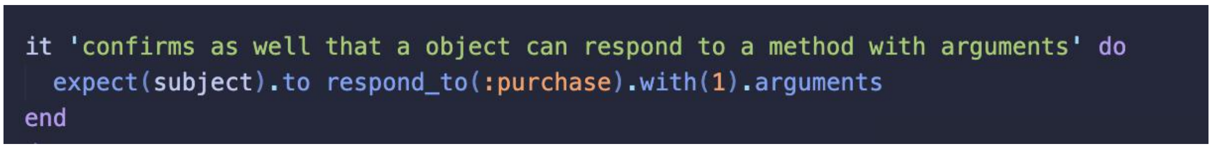
It checks that an object has a method

Other rspec methods are more about what a method returns or its implementation, this one is more concern about if an object can respond to a method.



```
it 'confirms that an object can respond to a method' do
  expect(subject).to respond_to(:drink)
  expect(subject).to respond_to(:drink, :discard, :purchase)
end
```

You can be more specific and see it it responds with certain quantity of parameters

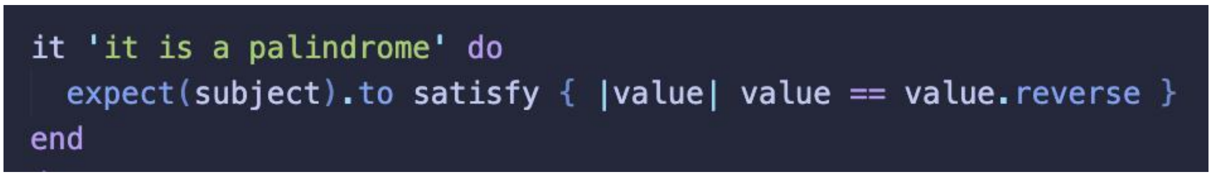


```
it 'confirms as well that a object can respond to a method with arguments' do
  expect(subject).to respond_to(:purchase).with(1).arguments
end
```

Polymorphism:
Comes from the word shape and many. It does not matter what object (shape it has), what it matters is that this object behaves as other objects.

47. satisfy Matcher

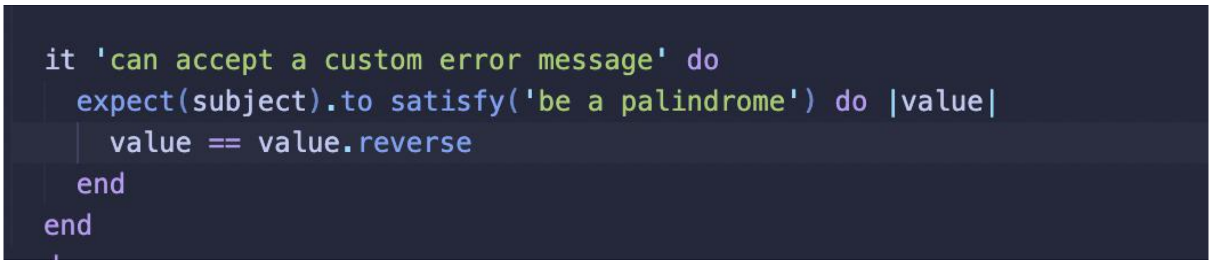
It allows us to have something similar as a custom expectation.
You pass a block and if the block returns true then it passes.



```ruby
it 'it is a palindrome' do
  expect(subject).to satisfy { |value| value == value.reverse }
end
```

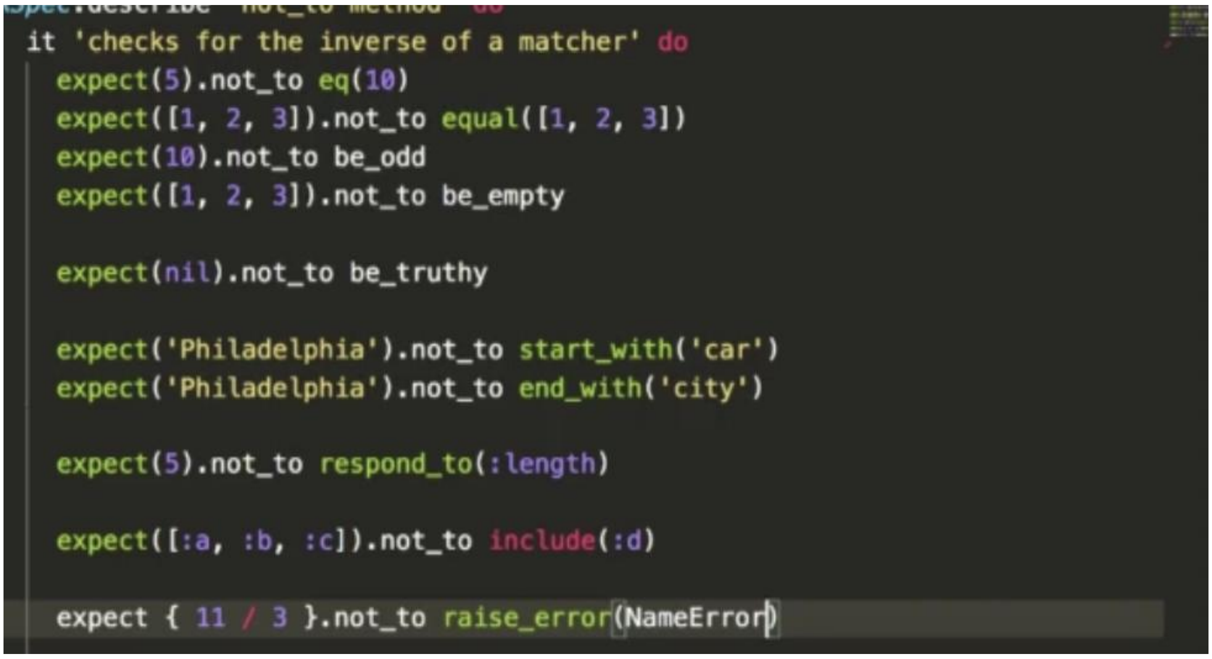
The problem is that the error this launches is not user friendly, a product manager will not understand it very well.

Add custom error message



```ruby
it 'can accept a custom error message' do
  expect(subject).to satisfy('be a palindrome') do |value|
    value == value.reverse
  end
end
```

48. not_to Method

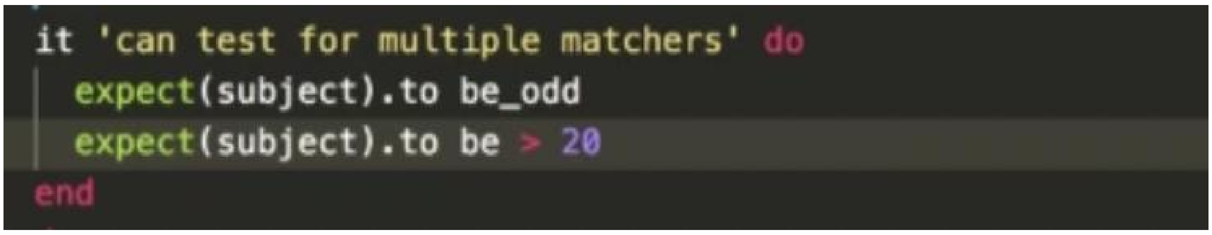


```ruby
it 'checks for the inverse of a matcher' do
  expect(5).not_to eq(10)
  expect([1, 2, 3]).not_to equal([1, 2, 3])
  expect(10).not_to be_odd
  expect([1, 2, 3]).not_to be_empty

  expect(nil).not_to be_truthy

  expect('Philadelphia').not_to start_with('car')
  expect('Philadelphia').not_to end_with('city')

  expect(5).not_to respond_to(:length)

  expect([:a, :b, :c]).not_to include(:d)

  expect { 11 / 3 }.not_to raise_error(NameError)
```

You can choose to use to or not_to, it depends on you, choose the one that better adapts to the context you are describing.

49. Compound Expectations

Usually if you want to check that two conditions are valid at the same time you can write:

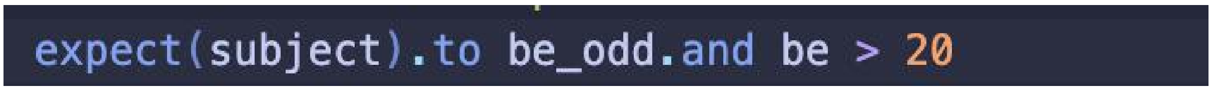


```ruby
it 'can test for multiple matchers' do
  expect(subject).to be_odd
  expect(subject).to be > 20
end
```

This is valid, but you can combine them on a single line.

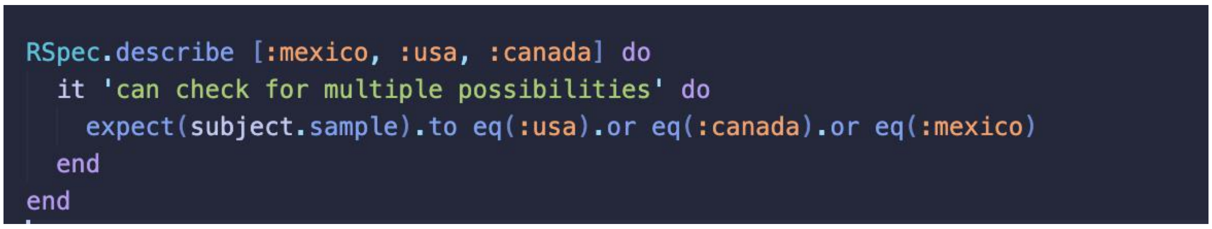
And

We can add as many ands as we want



```ruby
expect(subject).to be_odd.and be > 20
```

Or



```ruby
RSpec.describe [:mexico, :usa, :canada] do
  it 'can check for multiple possibilities' do
    expect(subject.sample).to eq(:usa).or eq(:canada).or eq(:mexico)
  end
end
```