cheat sheets.

$ cheat jasmine

```
Jasmine Javascript BDD Framework
http://github.com/pivotal/jasmine
http://groups.google.com/group/jasmine-js

Specs
===================
it('should be a test', function () {
  var foo = 0;
  foo++;
});

Expectations
===================

it('should be a test', function () {
  var foo = 0;            // set up the world
  foo++;                  // call your application code

  expect(foo).toEqual(1); // passes because foo == 1
});


Matchers
===================

expect(x).toEqual(y); // compares objects or primitives x and y and passes if
they are equivalent

expect(x).toMatch(pattern); // compares x to string or regular expression
pattern and passes if they match

expect(x).toBeDefined(); // passes if x is not undefined

expect(x).toBeNull(); // passes if x is null

expect(x).toBeTruthy(); // passes if x evaluates to true

expect(x).toBeFalsy(); // passes if x evaluates to false

expect(x).toContain(y); // passes if array or string x contains y

Every matcher's criteria can be inverted by prepending .not:

expect(x).not.toEqual(y); // compares objects or primitives x and y and passes
if they are not equivalent

Suites
===================
// Specs are grouped in Suites. Suites are defined using the global describe()
function:
describe('One suite', function () {
  it('has a test', function () {
    ...
  });

  it('has another test', function () {
    ...
  });
});
```

```
beforeEach
===================
var runnerWideFoo = [];

beforeEach(function () {
  runnerWideFoo.push('runner');
});

describe('some suite', function () {

  beforeEach(function () {
    runnerWideFoo.push('suite');
  });

  it('should equal bar', function () {
    expect(runnerWideFoo).toEqual(['runner', 'suite']);
  });
});


Single-spec After functions
===================
describe('some suite', function () {
  it(function () {
    var originalTitle = window.title;
    this.after(function() { window.title = originalTitle; });
    MyWindow.setTitle("new value");
    expect(window.title).toEqual("new value");
  });
});


Spies
===================
var Klass = function () {
};

var Klass.prototype.method = function (arg) {
  return arg;
};

var Klass.prototype.methodWithCallback = function (callback) {
  return callback('foo');
};

it('should spy on Klass#method') {
  spyOn(Klass, 'method');
  Klass.method('foo argument');

  expect(Klass.method).wasCalledWith('foo argument');
});

it('should spy on Klass#methodWithCallback') {
  var callback = jasmine.createSpy();
  Klass.methodWithCallback(callback);

  expect(callback).wasCalledWith('foo');
});


//Spies can be very useful for testing AJAX or other asynchronous behaviors that
take callbacks by faking the method firing an async call.
```

```
var Klass = function () {
};

var Klass.prototype.asyncMethod = function (callback) {
  someAsyncCall(callback);
};

...

it('should test async call') {
  spyOn(Klass, 'asyncMethod');
  var callback = jasmine.createSpy();

  Klass.asyncMethod(callback);
  expect(callback).wasNotCalled();

  var someResponseData = 'foo';
  Klass.asyncMethod.mostRecentCall.args[0](someResponseData);
  expect(callback).wasCalledWith(someResponseData);

});

// There are spy-specfic matchers that are very handy.
expect(x).wasCalled() // passes if x is a spy and was called

expect(x).wasCalledWith(arguments) // passes if x is a spy and was called with
the specified arguments

expect(x).wasNotCalled() // passes if x is a spy and was not called

expect(x).wasNotCalledWith(arguments) // passes if x is a spy and was not called
with the specified arguments


// Spies can be trained to respond in a variety of ways when invoked:
spyOn(x, 'method').andCallThrough() // spies on AND calls the original function
spied on

spyOn(x, 'method').andReturn(arguments) // returns passed arguments when spy is
called

spyOn(x, 'method').andThrow(exception) // throws passed exception when spy is
called

spyOn(x, 'method').andCallFake(function) // calls passed function when spy is
called


// Spies have some useful properties:
callCount // returns number of times spy was called

mostRecentCall.args // returns argument array from last call to spy.

argsForCall[i] // returns arguments array for call i to spy.

// Spies are automatically removed after each spec. They may be set in the
beforeEach function.


Asynchronous Specs
==================
it('should be a test', function () {
  runs(function () {
    this.foo = 0;
```

```
      var that = this;
      setTimeout(function () {
        that.foo++;
      }, 250);
    });

    runs(function () {
      this.expects(this.foo).toEqual(0);
    });

    waits(500);

    runs(function () {
      this.expects(this.foo).toEqual(1);
    });
  });
```