JS Testing Frameworks

CARL TEACHES YOU 5 FRAMEWORKS IN 5 MINS 😊

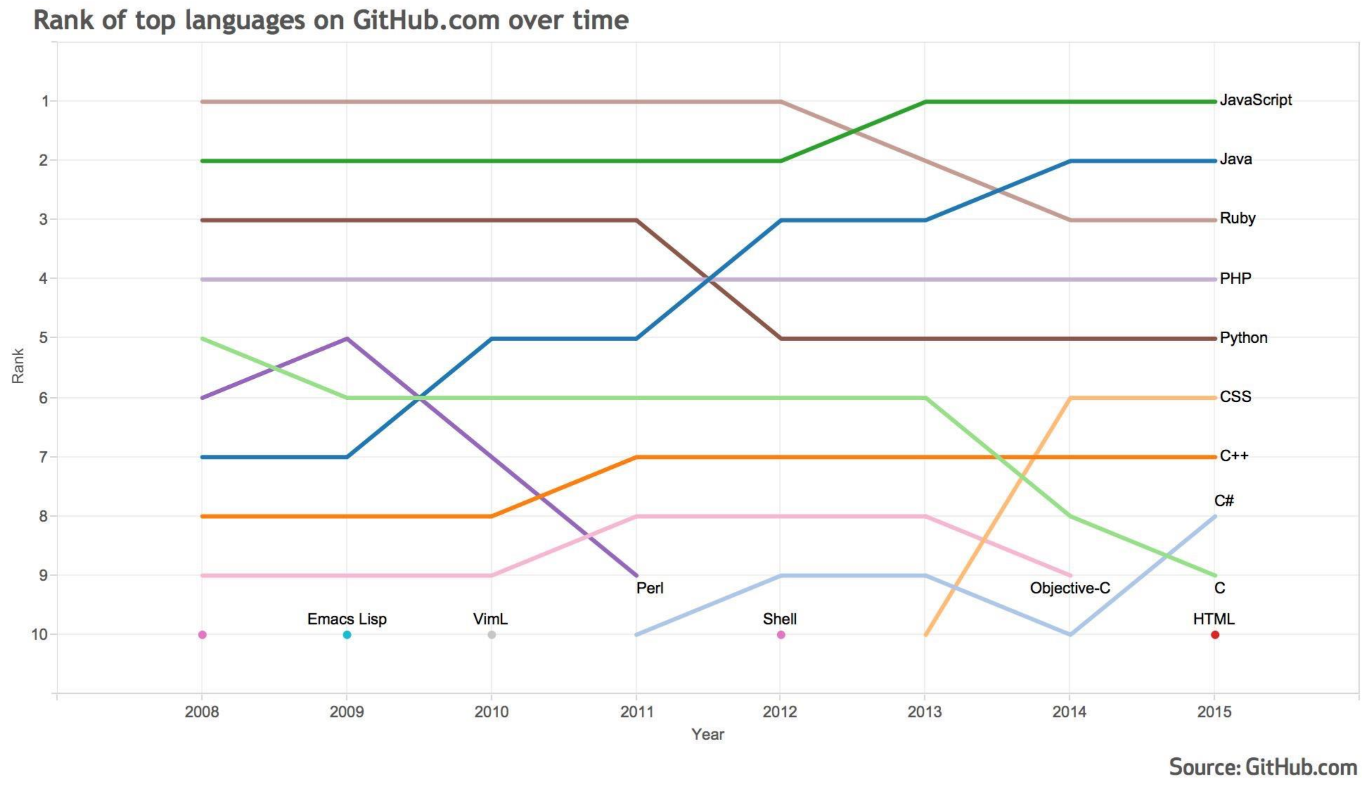
Goals

- An **easy** tour through some popular JavaScript testing frameworks/libraries.
- To arouse some interests in JavaScript.
- Thinking about test requirements in general.
- Covering:

yui Qu mocha Jasmine JsUnit

Why JavaScript?

1. It’s popular.
2. GNOME Shell uses it.
3. Actually most modern DEs do!



Poor Perl~

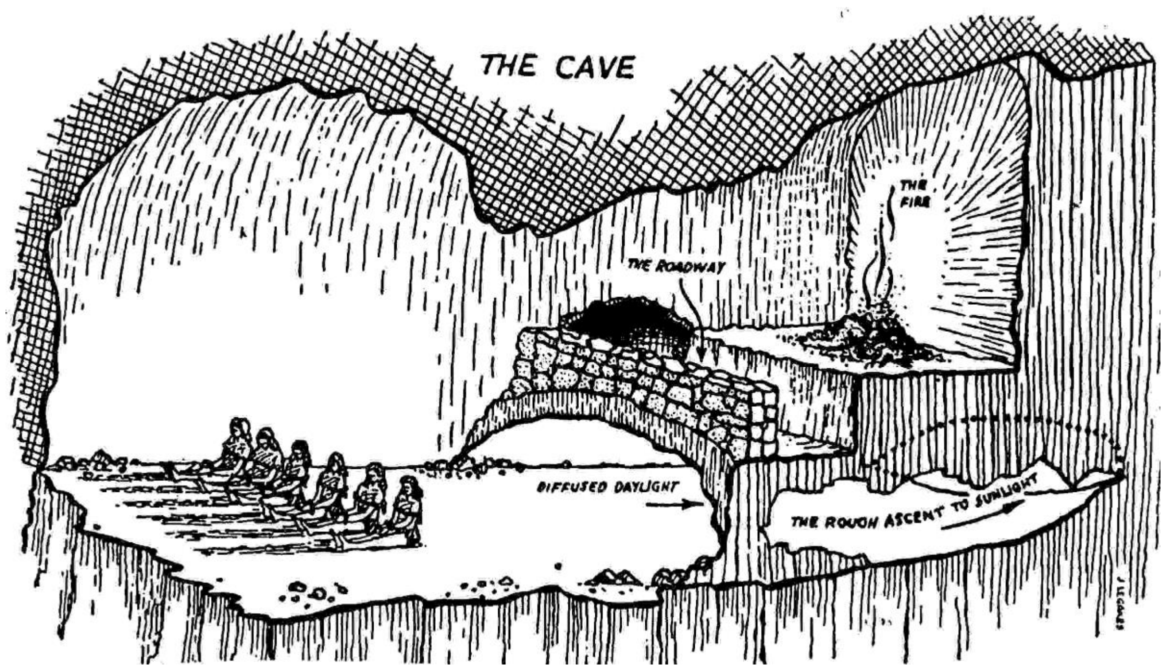
Status

- More than two dozens of JS test frameworks.
- GNOME Shell (or GJS) contains a copy of JSUnit.
- Pushing the use of JS on both server and client and even further (e.g. embedded system)

I DON'T USUALLY USE JAVASCRIPT.
BUT WHEN I DO, I USE IT FOR EVERYTHING.

Approach – Question Driven

- Assuming:
  - You’re a developer.
  - You want to test. You think it’s important.
  - What’re the requirements for your tools?
- The first question
  - What’s a test?



A Test is just “organized statements”

- Assertions.
- A single test case.
- Test suit: a set of test cases.
- Extras:
  - Setup, teardown.
  - Customization.
- These are what you learn the first, if not all.
- Syntax sugar indeed, essential,

  but don’t get addicted ;P

DSL, Interfaces

- **BDD** Style
  - Jasmine, (RSpec)
  - Mocha supports this by extension.
  - Focus on behavior.
  - Signatures**: describe**, **it, expect**

```
describe("Included matchers:", function() {

    it("The 'toBe' matcher compares with ===", function() {
        var a = 12;
        var b = a;

        expect(a).toBe(b);
        expect(a).not.toBe(null);
    });

    describe("The 'toEqual' matcher", function() {

        it("works for simple literals and variables", function() {
            var a = 12;
            expect(a).toEqual(12);
        });

        it("should work for objects", function() {
            var foo = {
                a: 12,
                b: 34
            };
            var bar = {
                a: 120,
                b: 340
            };
            expect(foo).not.toEqual(bar);
        });
    });
});
```

DSL, Interfaces, cont.

- **TDD/xUnit?** Style
  - JsUnit, YUI Test
  - Mocha supports this by extension.
  - Classic (first known to me through Junit)
  - Signatures: **testcase**, **setup**, **teardown**, **test***

```
var testCase = new Y.Test.Case({

    name: "TestCase Name",

    //---------------------------------------------
    // Setup and tear down
    //---------------------------------------------

    setUp : function () {
        this.data = { name : "Nicholas", age : 28 };
    },

    tearDown : function () {
        delete this.data;
    },

    //---------------------------------------------
    // Tests
    //---------------------------------------------

    testName: function () {
        Y.Assert.areEqual("Nicholas", this.data.name, "Name should be 'Nicholas'");
    },

    testAge: function () {
        Y.Assert.areEqual(28, this.data.age, "Age should be 28");
    }
});
```

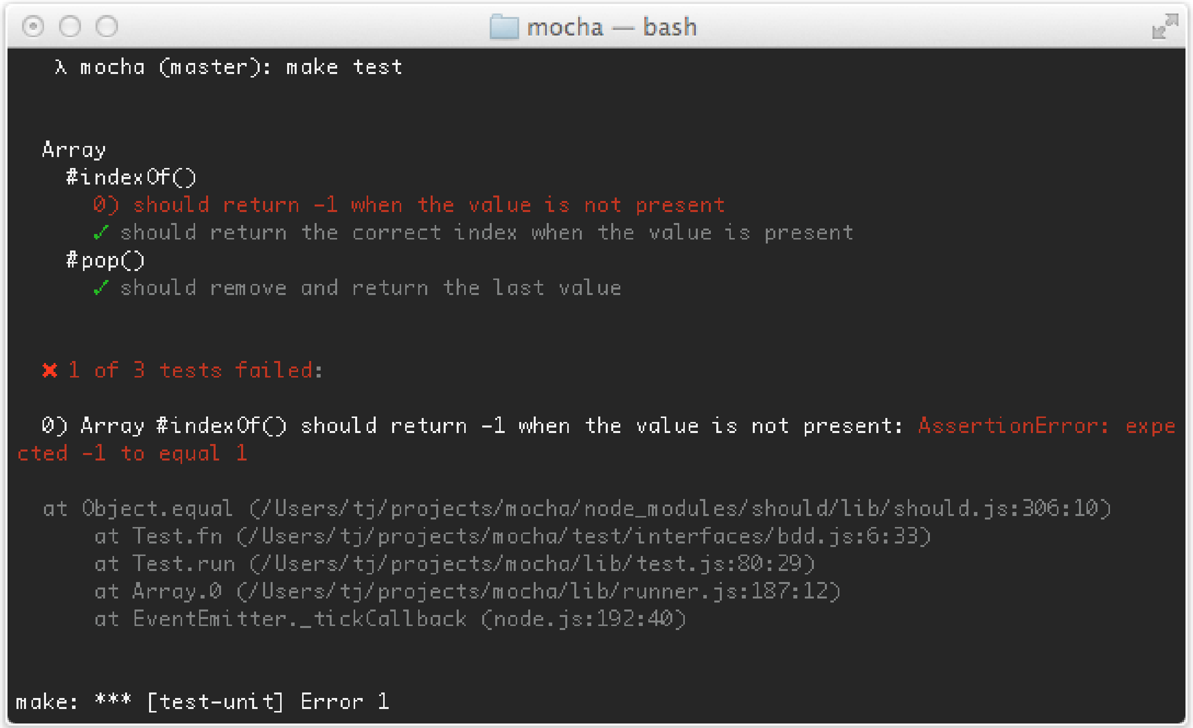
DSL, Interfaces, cont.

- **QUnit** Style
  - QUnit
  - Mocha supports this by extension.
  - A “flat” look when defining test suits.
  - Keywords: **module**, **test**

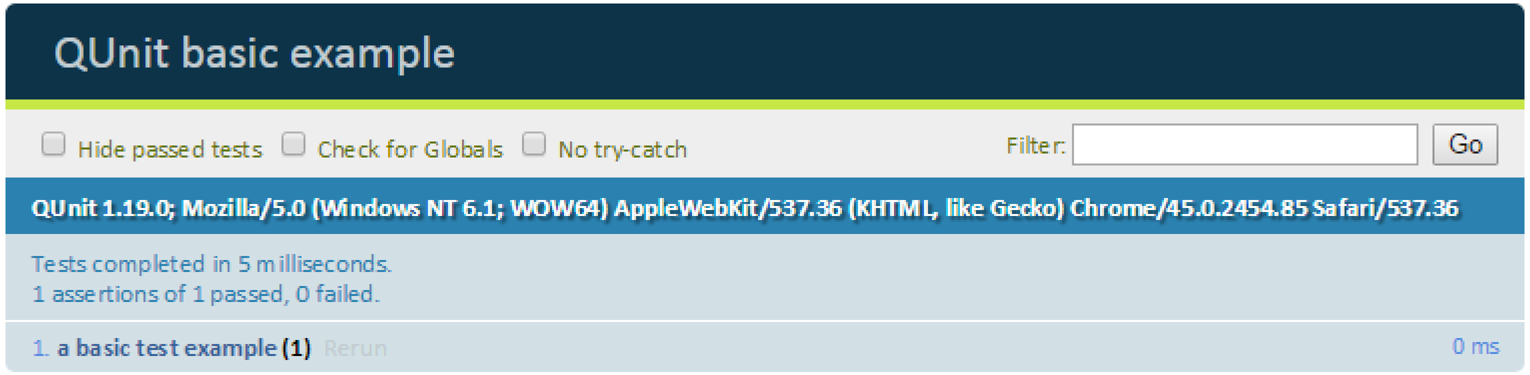
```
QUnit.module( "module", {

    beforeEach: function( assert ) {
        assert.ok( true, "one extra assert per test" );
    },

    afterEach: function( assert ) {
        assert.ok( true, "and one extra assert after each test" );
    }
});

QUnit.test( "test with beforeEach and afterEach", function() {
    assert.expect( 2 );
});
```

The things of tools

- ❑ Test runners.
- ❑ Reporters.
- ❑ Intuitive stuff, let's skim it through.



```
 λ mocha (master): make test


  Array
    #indexOf()
      0) should return -1 when the value is not present
      ✓ should return the correct index when the value is present
    #pop()
      ✓ should remove and return the last value


  ✖ 1 of 3 tests failed:

  0) Array #indexOf() should return -1 when the value is not present: AssertionError: expe
cted -1 to equal 1

  at Object.equal (/Users/tj/projects/mocha/node_modules/should/lib/should.js:306:10)
      at Test.fn (/Users/tj/projects/mocha/test/interfaces/bdd.js:6:33)
      at Test.run (/Users/tj/projects/mocha/lib/test.js:80:29)
      at Array.0 (/Users/tj/projects/mocha/lib/runner.js:187:12)
      at EventEmitter._tickCallback (node.js:192:40)


make: *** [test-unit] Error 1
```



QUnit basic example

QUnit 1.19.0; Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.85 Safari/537.36

Tests completed in 5 milliseconds.
1 assertions of 1 passed, 0 failed.

1. a basic test example (1) Rerun 0 ms

The order – welcome async

- Run and get it, fine. But what if it runs asynchronously?
  - Particularly common in JS as AJAX is a standard and popular technique.
- In general, you just pass callbacks around and set a timeout.
- In JS, since ES6, you have “promises” and “generators”.

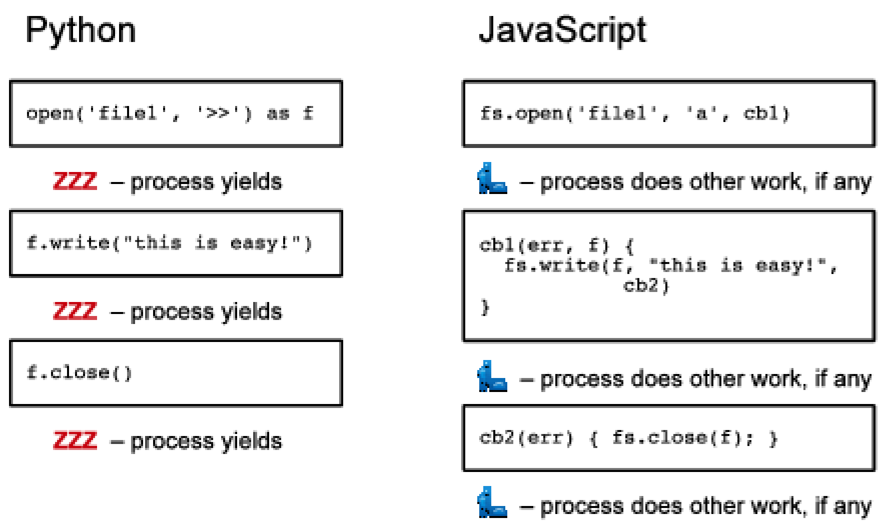
Async test with callbacks

- Callback: Standard Practice.
- Only minor differences among different frameworks.
- Use **Mocha**’s examples

```
describe('User', function() {
    describe('#save()', function() {
        it('should save without error', function(done) {
            var user = new User('Luna');
            user.save(function(err) {
                if (err) throw err;
                done();
            });
        });
    });
});
```

Async test with promises

- The **callback hell**
- No explanation. Just read some code


Python
JavaScript
ZZZ – process yields
ZZZ – process yields
ZZZ – process yields
– process does other work, if any
– process does other work, if any
– process does other work, if any

```
open('file1', '>>') as f
```

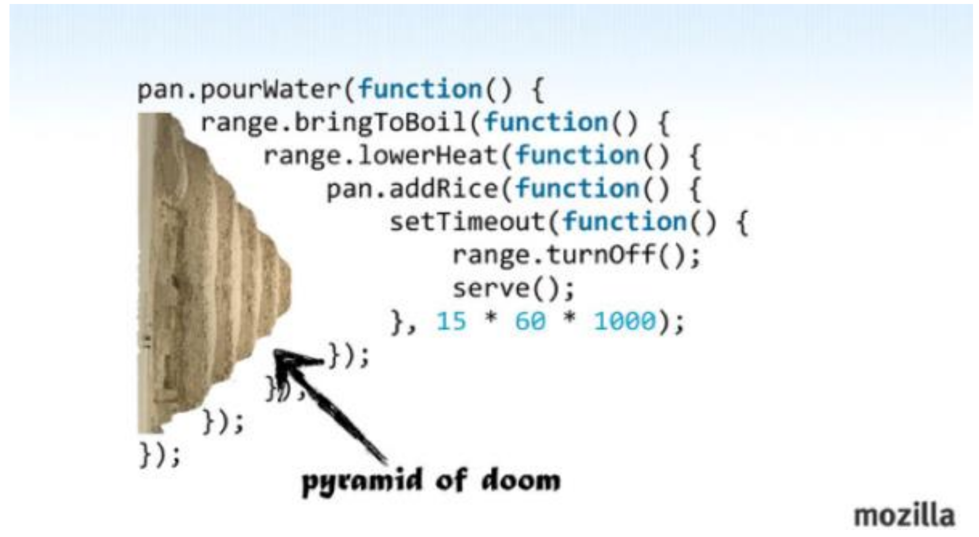
```
f.write("this is easy!")
```

```
f.close()
```

```
fs.open('file1', 'a', cb1)
```

```
cb1(err, f) {
  fs.write(f, "this is easy!",
               cb2)
}
```

```
cb2(err) { fs.close(f); }
```



```
pan.pourWater(function() {
    range.bringToBoil(function() {
        range.lowerHeat(function() {
            pan.addRice(function() {
                setTimeout(function() {
                    range.turnOff();
                    serve();
                }, 15 * 60 * 1000);
            });
        });
    });
});
```

Async test with promises, cont.

- Only **Mocha** has native promise support yet.

```
beforeEach(function() {
    return db.clear()
        .then(function() {
            return db.save([tobi, loki, jane]);
        });
});

describe('#find()', function() {
    it('respond with matching records', function() {
        return db.find({ type: 'User' }).should.eventually.have.length(3);
    });
});
```

Here comes the “Engineering”

- To be honest, you need more support from the framework.

  (Theoretically, the previous requirements make “perfect” test framework)
- Environment Integration
  - Server(nodejs), browsers support
    - custom servers, slave browsers, native DOM support and etc.
  - The lack of tests of GNOME Shell can be attributed to the lack of integration.
  - Mocks, stubs, emulators and etc.
- Hard to say more about this topic here.

Takeaway

- ❑ Try some JavaScript?
- ❑ Write your own test framework?
- ❑ A better understanding of how to pick the right test frameworks.
- ❑ Question Driven is cool!

Thx~