“JavaScript” means Different things to Different People
<form name='FORM'>
<table border cellpadding=3>
<tr>
<td><nobr>radius: <input name='Circle_radius' size=4/></nobr></td>
<td><input type='button' onclick='document.FORM.submit();' value='calculate'/></td>
<td align='right' bgcolor='#AACCFF'>
<nobr>circumference: <input name='Circle_circumference' size=9/></nobr><br>
<nobr>area: <input name='Circle_area' size=9/></nobr></td>
</tr>
</table>
</form>
var http = require('http');
http.createServer(function (req, res) {
    res.writeHead(200, {'Content-Type': 'text/plain'});
    res.end('Hello World\n');
}).listen(8124, "127.0.0.1");
console.log('Server running at http://127.0.0.1:8124/');
**DIVE INTO HTML 5**

**BY**

**MARK PILGRIM**

**WITH ILLUSTRATIONS FROM THE PUBLIC DOMAIN**

---

Dive Into HTML 5 seeks to elaborate on a hand-picked selection of features from the HTML5 specification and other fine standards. I shall publish Drafts periodically, as time permits. Please send feedback. The final manuscript will be published on paper by O'Reilly, under the Google Press imprint. Pre-order the printed Work and be the first in your community to receive it. The Work shall remain online under the CC-BY-3.0 License.
ChessFlock
ejabberd: XMPP server written in Erlang

kibosh: XMPP BOSH connection manager written in Ruby

strophejs: XMPP BOSH client written in JavaScript

dramatis: actor library written in JavaScript/Ruby/Python

chessflock: chess client/server written in JavaScript/HTML/SVG
Caveats
Incomplete
Low test coverage
Egregious stupidity
Dramatis testbed
“throw one away”
Test Driven Development
s/Test/Behavior/
Test Driven Development
Test Driven Development
Test Driven Development
Test Driven Development
Write the tests first
Red Green Refactor
Flavors of tests
Unit
Irreducible
Integration

Multiple units

Maybe multiple languages
Unit
Monolingual (JavaScript) Integration
Developer focused
What makes JavaScript so special?
Standard Library

or lack thereof
Object
Array
Function
Date
String
Regex
Asynchronous
no wait
no sleep
not threads
Asynchronous
no wait
no sleep
no threads

setTimeout
Asynchronous
  no wait
  no sleep
  no threads

window.setTimeout
Asynchronous
no wait
no sleep
no threads

`window.setTimeout`
Asynchronous
no wait
no sleep
no Threads

window.setTimeout

I/O
Asynchronous
no wait
no sleep
no threads

window.setTimeout
I/O
Packaging
Asynchronous
no wait
no sleep
no threads

window.setTimeout
I/O
Packaging
Need more than this to run tests

Object
Array
Function
Date
String
Regex
Beyond ES5
CommonJS
“HTML5”
window
window.setTimeout
window.setTimeout

Timers / events / event loops
window.XMLHttpRequest
HTML5 implementations
Browsers, of course but also ...
HTMLUnit
Java
Rhino
Env.js

“mostly” JavaScript

Rhino/SpiderMonkey
JavaScript TDD/BDD vs ...
Dynamically typed

Loose scoping

No protection levels
Highly asynchronous
Lots of callbacks
Lots of async callbacks
JavaScript TDD and BDD Frameworks
But first ...
JSLint
JavaScript TDD: qunit
test("module without setup/teardown (default)", function() {
  expect(1);
  ok(true);
});
test("module without setup/teardown (default)", function() {
    expect(1);
    ok(true);
});
test("module without setup/teardown (default)", function() {
  expect(1);
  ok(true);
});
test("runnable callbacks are run later with timeout of 0", function() {
    expect(2);
    var occurred = 0;
    setTimeout(function() {
        occurred = Date.now();
    }, 0);
    ok( occurred === 0, "Timeout callback was not executed immediately" );
    setTimeout(function() {
        ok( occurred !== 0, "Timeout callback executed" );
        start();
    }, 100);
    stop();
});
test("Runnable callbacks are run later with timeout of 0", function() {
    expect(2);
    var occurred = 0;
    setTimeout(function() {
        occurred = Date.now();
    }, 0);
    ok( occurred === 0, "Timeout callback was not executed immediately");
    setTimeout(function() {
        ok( occurred !== 0, "Timeout callback executed");
    }, 100);
    start();
    stop();
});

stop()
test("runnable callbacks are run later with timeout of 0", function() {
    expect(2);
    var occurred = 0;
    setTimeout(function() {
        occurred = Date.now();
    }, 0);
    ok( occurred === 0, "Timeout callback was not executed immediately" );
    setTimeout(function() {
        ok( occurred !== 0, "Timeout callback executed" );
        start();
    }, 100);
    stop();
});
```javascript
function() {
    expect(2);
    var occurred = 0;
    setTimeout(function()
        occurred = Date.now();
    }, 0);
    ok( occurred === 0, "Timeout callback was not executed immediately" );
    setTimeout(function()
        ok( occurred !== 0, "Timeout callback executed" );
        start();
    }, 100);
    stop();
});
```
test("runnable callbacks are run later with timeout of 0", function() {
    expect(2);
    var occurred = 0;
    setTimeout(function() {
        occurred = Date.now();
    }, 0);
    ok( occurred === 0, "Timeout callback was not executed immediately" );
    setTimeout(function() {
        ok( occurred !== 0, "Timeout callback executed" );
        start();
    }, 100);
    stop();
});

complete()
test("runnable callbacks are run later with timeout of 0", function() {
    expect(2);
    var occurred = 0;
    setTimeout(function(){
        occurred = Date.now();
    }, 0);
    ok( occurred === 0, "Timeout callback was not executed immediately" );
    setTimeout(function(){
        ok( occurred !== 0, "Timeout callback executed" );
        start();
    }, 100);
    stop();
});
What if start() is never called?
JavaScript BDD: Jasmine
it("should return the game to the players", function() {
    var tom = { player_name: function() { return "tom"; } }
    var jerry = { player_name: function() { return "tom"; } }
    var toms_game;
    var jerrys_game;
    this.server.join(tom, function(game) {
        toms_game = game;
        if (jerrys_game) {
            expect(toms_game).toBe(jerrys_game);
            complete();
        }
    });
    this.server.join(jerry, function(game) {
        jerrys_game = game;
        if (toms_game) {
            expect(toms_game).toBe(jerrys_game);
            complete();
        }
    });
    incomplete();
});
Ready, Set, Go ...
Uh ...
Fixtures
undefined = "diz be undefined yo";
</script>

<script type="text/javascript" src="../src/html/jasmine.css"></script>
<script type="text/javascript">/
// yes, really keep this here to keep us honest, but only for jasmine's own runner! [xw]
</script>

<script type="text/javascript" src="../src/base.js"></script>
<script type="text/javascript" src="../src/util.js"></script>
<script type="text/javascript" src="../src/Runner.js"></script>
<script type="text/javascript" src="../src/Suite.js"></script>
<script type="text/javascript" src="../src/Spec.js"></script>
<script type="text/javascript" src="../src/Env.js"></script>
<script type="text/javascript" src="../src/WaitsBlock.js"></script>
<script type="text/javascript" src="../src/WaitsForBlock.js"></script>
<script type="text/javascript" src="../src/mock-timeout.js"></script>
<script type="text/javascript">
(function() {
    var jasmineEnv = jasmine.getEnv();
    jasmineEnv.updateInterval = 1000;

    var trivialReporter = new jasmine.TrivialReporter();

    jasmineEnv.addReporter(trivialReporter);

    jasmineEnv.specFilter = function(spec) {
        return trivialReporter.specFilter(spec);
    };

    window.onload = function() {
        jasmineEnv.execute();
    };

})();
</script>

<link href="../src/html/jasmine.css" rel="stylesheet"/>
</head>

<body>
</body>
</html>
Test Runners:
What, Where, How, When, Why
Test Runners:
What, Where, How, When
Without tests, code won’t work
Without tests, code won’t work

If writing tests is hard, they won’t get written
Without tests, code won’t work

If writing tests is hard, they won’t get written

If running tests is hard, they won’t get run
Without tests, code won’t work

If writing tests is hard, they won’t get written

If running tests is hard, they won’t get run

If tests don’t get run, they won’t get written
Without tests, code won’t work

If writing tests is hard, they won’t get written

If running tests is hard, they won’t get run

If tests don’t get run, they won’t get written

Without tests, code won’t work
Make the simple things easy and the hard things possible
DOM vs non-DOM

Synch vs Asynch
Make the hard things possible
Make the hard things possible and everything else equally hard
Write a test
Switch to browser
Load a test page
Find the/a failure
Switch to editor
Write code
Switch to browser
Hit refresh
Find the/a success

Lather, rinse, repeat
This sucks
Let me count the ways
Load what?
From where?
How, i.e., with what?
When?
What: pick your fixture(s)

Where: run some server somewhere

How: with the browser on my desktop (repeat as necessary)

When: when I feel like it
Did I mention this sucks?
What: I hate fixtures
They’re painful to setup

They don’t reflect the real world
In vitro
In situ
In vivo
Jazz: a test runner
Jazz

Supports Jasmine and qunit

Supports browsers (Chrome, Firefox)
Supports env.js (Ruby/SpiderMonkey)

Can run fixture-less tests

Integrates with wake
Wake: make/rake + autotest/watchr
Save a file, run a test
Save a file, run tests
Understands dependences

Integrated with jazz/envjs
Records successes/failures

Re-executes:
  Changed files until success
  Failed tests until success
  All tests
Plugin Architecture

Matching source files
Result files

Dependences
Success/failure
Plugins

haml/sass
jazz
jslintrb
shell
graphics (inkscape, batik, rsvg)
cache_manifest
qunit: http://github.com/jquery/qunit
Jasmine: http://github.com/pivotal/jasmine
  my fork: http://github.com/smparkes/jasmine
jslintrb: http://github.com/smparkes/jslintrb
jazz: http://github.com/smparkes/jazz
wake: http://github.com/smparkes/wake
ttt: http://github.com/smparkes/ttt
chessflock: http://github.com/smparkes/chessflock