coreservlets.com – custom onsite training

JavaScript Unit Testing with JsUnit

For additional materials, please see http://www.coreservlets.com/. The JavaScript tutorial section contains complete source code for all examples in the entire tutorial series, plus exercises and exercise solutions for each topic.

coreservlets.com – custom onsite training

For customized training related to JavaScript or Java, email hall@coreservlets.com

Marty is also available for consulting and development support

Taught by lead author of *Core Servlets & JSP*, co-author of *Core JSF* (4th Ed), and this tutorial.

Available at public venues, or custom versions can be held on-site at your organization.

- **Courses developed and taught by Marty Hall**
  - JavaScript, jQuery, Ext JS, JSF 2.3, PrimeFaces, Java 8 programming, Spring Framework, Spring MVC, Android, GWT, custom mix of topics
  - Courses available in any state or country.
  - Maryland/DC companies can also choose afternoon/evening courses.
- **Courses developed and taught by coreservlets.com experts (edited by Marty)**
  - Hadoop, Hibernate/JPA, HTML5, RESTful Web Services

Contact hall@coreservlets.com for details

Topics in This Section

- **JsUnit overview**
  - Summary
- **Installing and using JsUnit**
  - Details
- **Available assertion functions**
  - assertTrue, assertEquals, assertNull, assertUndefined, assertNaN (and opposites)
- **Examples**
  - Simple and more realistic
- **Unit testing jQuery DOM manipulation**
  - Comparing HTML values
- **Unit testing jQuery Ajax**
  - Using setUpPage to guarantee aynchronous code finishes before tests are performed

JsUnit Overview

Motivation

- **Unit testing in general**
  - Testing individual methods or small pieces of functionality. Testing overall behavior often not sufficient because not all code cases are used in integrated tests.
  - Run on multiple browsers to verify consistent results
  - Whenever you modify code, rerun the test cases to verify you are still getting same answer
- **JsUnit in particular**
  - Most widely used unit testing framework in JavaScript world
    - Not the only unit testing framework, or even necessarily the best
    - Even so, due to its popularity, most JavaScript developers start with JsUnit first
  - Easy to learn basics
  - Very similar to classic-style JUnit from the Java world
    - However, compared to modern JUnit, JsUnit is weaker and much more poorly documented

Initial Debugging vs. Long-Term Unit Tests

- **Initial debugging: interactive calls**
  - When you first write your code, it is simple (and recommended!) to go to Firebug or the Chrome developer tools console and type in function calls interactively and see if they output what you expect. You would type in everything here that you would ever think to put into unit tests, and probably a lot more.
    - It would be time consuming and unrealistic to use unit tests for this
- **Later verification: unit tests**
  - After code has been debugged, you want to verify that it still works when you *later* make changes to it. So, save some of the most important tests cases into unit tests.
    - Rerun the unit tests when you make changes to the code
- **Set initial expectations low**
  - Most developers do *no* unit testing whatsoever! Anything is better than nothing. So, start by creating and saving some basic unit tests to your most important functions.
    - You can gradually add more tests later

Summary of Steps

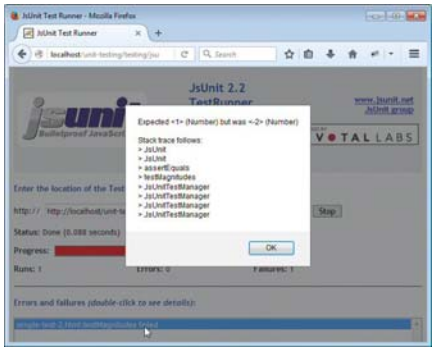
- **Download JsUnit**
  - http://sourceforge.net/projects/jsunit/files/
- **Copy the jsunit folder into your application**
  - your-app/testing/jsunit/...
- **Load jsunit/app/jsUnitCore.js in an HTML test page**
  - <script src="jsunit/app/jsUnitCore.js"></script>
- **In that page, make one or more functions called testBlah**
  - <script>function testMyCode() { ... }</script>
- **Use assertEquals, assertTrue, etc. in the testBlah function**
  - function testMyCode() { assertTrue(isPrime(13)); }
- **Load testRunner.html in a browser**
  - http://your-host/your-project/jsunit/testRunner.html
- **Enter address of your test page and click Run**
  - Enter http://your-host/your-project/your-test-page.html

Quick Example: simple-test-1.html

```
...
<script src="jsunit/app/jsUnitCore.js"></script>
...
<script>
function smallerMagnitude(num1, num2) {
  return(Math.min(Math.abs(num1), Math.abs(num2)));
}

function testMagnitudes() {
  assertEquals(1, smallerMagnitude(1, 2));
  assertEquals(1, smallerMagnitude(-1, -2));
  assertNaN(smallerMagnitude("bad", "values"));
  assertNaN(smallerMagnitude(1));
}
</script>...
```

Quick Example: simple-test-2.html

```
...
<script src="jsunit/app/jsUnitCore.js"></script>
...
<script>
function smallerMagnitude(num1, num2) {
  return(Math.min(num1, num2)); // Forgot absolute value
}

function testMagnitudes() {
  assertEquals(1, smallerMagnitude(1, 2));
  assertEquals(1, smallerMagnitude(-1, -2));
  assertNaN(smallerMagnitude("bad", "values"));
  assertNaN(smallerMagnitude(1));
}
</script>...
```



References

- **Home page**
  - http://www.jsunit.net/
- **Download**
  - http://sourceforge.net/projects/jsunit/files/
- **Documentation**
  - http://www.jsunit.net/documentation/

Installing and Using JsUnit: Details

Summary of Steps

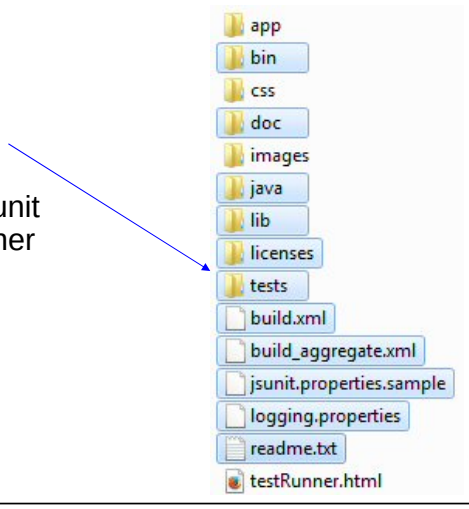
- **Download JsUnit**
  - http://sourceforge.net/projects/jsunit/files/
- **Copy the jsunit folder into your application**
  - your-app/testing/jsunit/...
- **Load jsunit/app/jsUnitCore.js in an HTML test page**
  - <script src="jsunit/app/jsUnitCore.js"></script>
- **In that page, make one or more functions called testBlah**
  - <script>function testMyCode() { ... }</script>
- **Use assertEquals, assertTrue, etc. in the testBlah function**
  - function testMyCode() { assertTrue(isPrime(13)); }
- **Load testRunner.html in a browser**
  - http://your-host/your-project/jsunit/testRunner.html
- **Enter address of your test page and click Run**
  - Enter http://your-host/your-project/your-test-page.html

Download JsUnit

- **Download zip file**
  - Start at http://www.jsunit.net/ and click Download at the top right
  - This points to http://sourceforge.net/projects/jsunit/files/
  - Click on entry near top next to prompt that says “Looking for the latest version?”
  - Downloading this will result in something similar to jsunit2_2.zip
- **Unzip into location of your choice**
  - Resulting in folder named jsunit

Copy the jsunit Folder Into Your App

- **Put folder you obtained in previous slide inside your app**
  - Resulting in something like your-app/testing/jsunit/...
- **Optional: delete extraneous folders and files**
  - Required folders: app, css, images.
  - Required top-level file: testRunner.html
  - To keep your app simpler and easier to understand, the rest can be deleted
    - Your unit tests will run fine if you simply copy the jsunit folder exactly as is, but most developers like a cleaner folder structure



Copy jsunit Folder: In Real Application

- **Put jsunit in real application**
  - Advantage: you can run tests any time, without copying any files
  - Disadvantage: you should delete testing folder from deployed version
- **Your test pages are usually in a subfolder**
  - E.g., "testing"

- **Typical App Layout**
  - scripts/... (JavaScript code for main app)
  - css/... (CSS files for main app)
  - images/... (images for main app)
  - regular-application-file-1.html
  - regular-application-file-2.html
  - **testing/jsunit/app/...**
  - **testing/jsunit/css/...**
  - **testing/jsunit/images/...**
  - **testing/jsunit/testRunner.html**
  - testing/test-file-1.html
    - Will test JavaScript code from main app
  - testing/test-file-2.html
    - Will test JavaScript code from main app

Copy jsunit Folder: In Test Application

- **Put jsunit in test application**
  - Advantage: not for deployment, so no need to ever delete anything
  - Disadvantage: every time you want to test modified code, you have to copy JavaScript files from the real app into the test app
- **Your test pages are usually in the top level**
  - E.g., no "testing" subfolder

- **Typical App Layout**
  - scripts/...
    - (Code copied from the main app)
  - css/... (often missing)
    - Only if your unit testing will involve CSS
  - images/... (often missing)
    - Only if your unit testing will involve images
  - **jsunit/app/...**
  - **jsunit/css/...**
  - **jsunit/images/...**
  - **jsunit/testRunner.html**
  - test-file-1.html
    - Will test JavaScript code that was copied from main app
  - test-file-2.html
    - Will test JavaScript code that was copied from main app

Load jsunit/app/jsUnitCore.js in an HTML Test Page

- **Create an HTML page solely for testing**
  - Depending on which approach from the previous two slides you use, this could be in a testing-specific subfolder, or in the top level of your app
- **Load the jsUnitCore.js file**
  - Which is probably in a subfolder relative to the HTML test file
- **Typical HTML**

```
<!DOCTYPE html>
<html>
<head>...
<script src="jsunit/app/jsUnitCore.js"></script>
</head>
<body>...</body>
</html>
```


Make One or More Functions Called test*Blah*

- **The name of the function literally starts with "test"**
  - JsUnit will automatically find these functions and run them when the HTML test page is loaded in the JsUnit test runner
- **But, you can override this convention**
  - Define a function called exposeTestFunctionNames that returns a string array referring to the test function names
- **Example**

```
...
<body>
<script>
function testMyCode() { ... }
</script>
</body>
</html>
```

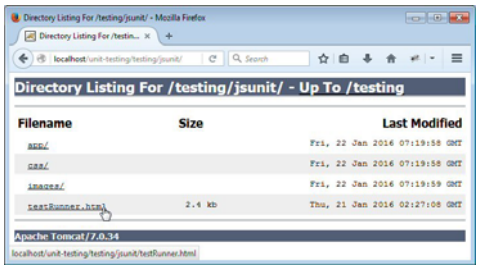
Use assertEquals, assertTrue, etc.

```
<script>
function isPrime(number) {
  // Return true if number is prime, false otherwise
}
function nextPrime(number) {
  // Return first prime higher than the given number
}

function testMyCode() {
  assertTrue(isPrime(13));
  assertFalse(isPrime(14));
  assertEquals(9973, nextPrime(9970));
}
</script>
```

A list of the available assertion functions and their meanings is in the next subsection. But the meaning of these is moderately obvious:
- assertTrue: I think this is true; tell me if I am wrong
- assertFalse: I think this is false; tell me if I am wrong
- assertEquals: I think the two values are equal; tell me if I am wrong

Load testRunner.html in a Browser

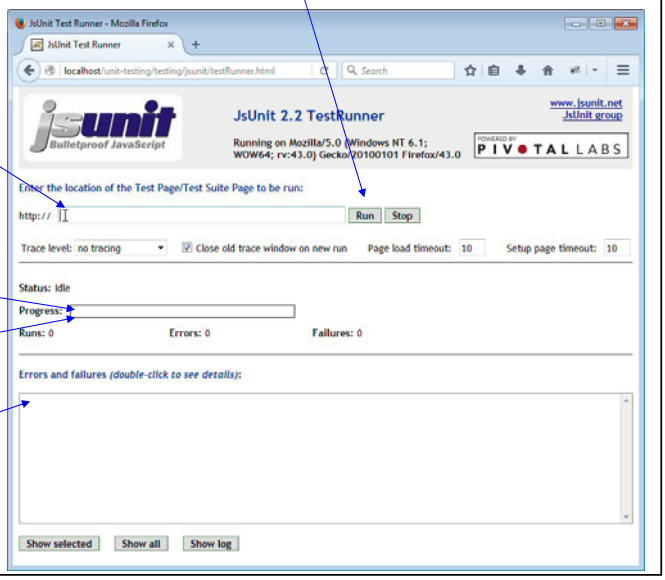
- **Deploy your app to server and load jsunit/testRunner.html**
  - If you use the approach with separate testing folder, URL would be similar to
    `http://your-host/your-app/testing/jsunit/testRunner.html`
    - Since you normally test on a local server, this URL is likely to be
      `http://localhost/your-app/testing/jsunit/testRunner.html`
  - If you use the approach with a separate app just for testing, URL would be similar to
    `http://your-host/your-app/jsunit/testRunner.html`
    - Since you normally test on a local server, this URL is likely to be
      `http://localhost/your-app/jsunit/testRunner.html`
  - In either case, if your server has directory listings enabled, you can first enter the URL of the jsunit folder, then click on testRunner.html



Enter Address of Test Page and Click Run

Enter URL of your test page here. It will be something like http://localhost/your-app/testing/your-test-page.html

After entering the URL, press Run



- **If all the tests pass**
  - This will show a green bar
- **If any of the tests fail**
  - This will show a red bar
  - This will show message for the first failed test. Double click that message to get details on what went wrong.

Available Assertion Functions

Overview

- **Functions**
  - assertTrue (equivalent to just “assert”), assertFalse
  - assertEquals, assertNotEquals
  - assertNull, assertNotNull
  - assertUndefined, assertNotUndefined
  - assertNaN, assertNotNaN
  - fail

Available Assertion Functions

- **assertTrue(value)**
  - Claims that value evaluates to true.

```
assertTrue(isPrime(9973));
```

- **assertFalse(value)**
  - Claims that value evaluates to false. assertFalse(blah) is same as assertTrue(!blah)

```
assertFalse(isPrime(9971));
```

- **assertEquals(val1, val2)**
  - Claims that the two values are equal (==). Using assertEquals(a, b) is clearer and has better messages on failure than using assertTrue(a == b).

```
assertEquals(9973, nextPrime(9970));
```

- **assertNotEquals(val1, val2)**
  - Claims that the two values are not equal. Using assertNotEquals(a, b) is clearer and has better messages on failure than assertFalse(a == b).

```
assertNotEquals(slowestCar(cars), fastestCar(cars));
```

Available Assertion Functions (Continued)

- **assertNull(value)**
  - Claims that the value is == to null.

```
assertNull(middleName(personThoughtToLackMiddleName));
```

  - Many JavaScript programmers return null from functions that normally return objects, but when no valid object applies in the current situation. For example:
    - `firstPalindrome(["bob", "blob"]) → "bob"`
    - `firstPalindrome(["jill", "hill"]) → null`
- **assertNotNull(value)**
  - Claims that the value is non-null

```
assertNotNull(middleName(personThoughtToHaveMiddleName));
```


Available Assertion Functions (Continued)

- **assertUndefined(value)**
  - Claims that the value is undefined (never had a value)

```
var nums = [1, 2, 3];
assertUndefined(nums[3]);
```

  - Note that null and undefined are *not* the same. Both can be coerced to false in a conditional:

```
if(blah) { doThis(); }
```

    - doThis() will not be called if blah is null or undefined, but the test also fails if blah is false, "" (empty string), 0, or NaN. All of those are distinct values.
- **assertNotUndefined(value)**
  - Claims that the value is not undefined

```
var nums = [1, 2, 3];
assertNotUndefined(nums[2]);
```

Available Assertion Functions (Continued)

- **assertNaN(number)**
  - Claims that number is NaN ("Not a Number", which is actually a legal numeric value). It is difficult to do numerical comparisons, since all comparisons (==, <, >) to NaN return false. Even n == n returns false when n is NaN.

```
assertNaN(parseFloat("bad user data"));
```

  - Many JavaScript functions return NaN when they would normally return a number but no number applies in the current situation.
    - `parseInt("bogus") → NaN`
    - `parseFloat("bogus") → NaN`
    - `0/0 → NaN`
    - `Math.sqrt(-1) → NaN`
    - `Math.log(-1) → NaN`
- **assertNotNaN**
  - Claims that the number is anything except NaN.

```
assertNotNaN(parseFloat("2.3"));
```


Available Assertion Functions (Continued)

- **fail(comment)**
  - Causes the test to always fail.
  - Often used in a place in the code that you think is unreachable

```
fail("Error if this line is reached");
```

Order of Arguments for assertEquals

- **Put literal value first**
  - When comparing the output of your function to a literal value, it is customary to put the literal value first.
  - Yes
    ```
    assertEquals(2, smallestNum(someArray));
    ```
  - No
    ```
    assertEquals(smallestNum(someArray), 2);
    ```
- **Motivation: error messages**
  - Although obviously the two forms will succeed or fail in all the same situations, when the assertion fails, the error message for the first form is more clear:
    ```
    Expected <2> (Number) but was <7> (Number)
    ```
  - The error message for the second form would have been
    ```
    Expected <7> (Number) but was <2> (Number)
    ```


Comment Strings in Assertions

- **All assertions take a string as optional first argument**
  - assertTrue(value)
  - assertTrue("message", value)
  - assertEquals(val1, val2)
  - assertEquals("message", val1, val2)
  - assertNull(value)
  - assertNull("message", value)
  - Etc

Motivation: Unambiguous Messages

- **Example 1**
  - Assertions

```
assertEquals(2, smallestNum(shortArray));
assertEquals(2, smallestNum(longArray));
```

  - Ambiguous failure message

```
Expected <2> (Number) but was <7> (Number)
```

- **Example 2**
  - Assertions

```
assertEquals("Smallest in Short Array",
             2, smallestNum(shortArray));
assertEquals("Smallest in Long Array",
             2, smallestNum(longArray));
```

  - Unambiguous failure message

```
Smallest in Short Array
Expected <2> (Number) but was <7> (Number)
```

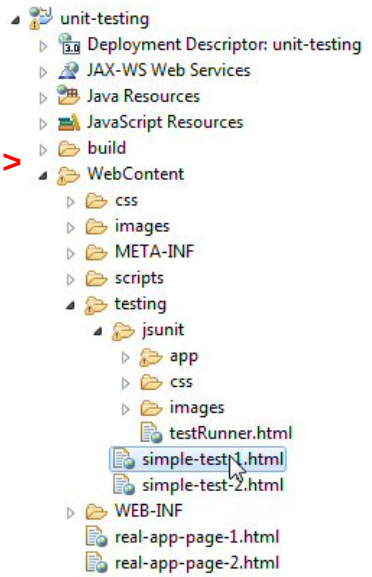

Simple Example

Overview

- **Function to be tested**
  - smallerMagnitude: should return the smaller of the two magnitudes (absolute values) of the numbers supplied
- **Application setup**
  - Put jsunit and testing pages into subfolder (testing) of existing application
- **Test page 1**
  - Tests with initial debugged version of the function.
    - All tests pass
- **Test page 2**
  - Tests with later refactored version of the function. Same tests as before, but whether they pass or not depends on whether the function still performs the same way.
    - Rigged to show failure

Test Page 1: Part 1

```
<!DOCTYPE html>
<html>
<head><title>Unit Testing with JsUnit</title>
<script src="jsunit/app/jsUnitCore.js"></script>
</head>
<body>
```



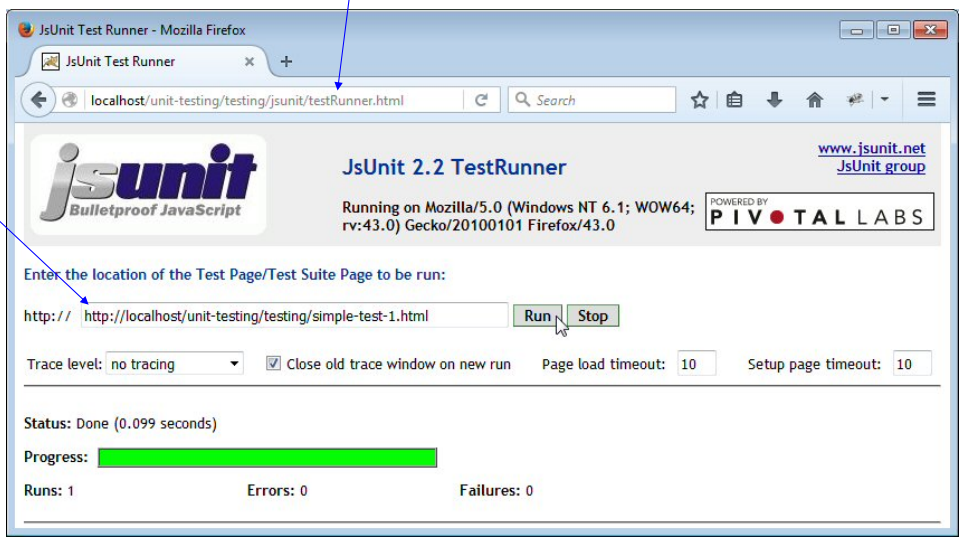
Test Page 1: Part 2

```
<script>
function smallerMagnitude(num1, num2) {
  return(Math.min(Math.abs(num1), Math.abs(num2)));
}

function testMagnitudes() {
  assertEquals("Testing with positive numbers"
               1, smallerMagnitude(1, 2));
  assertEquals("Testing with mixed numbers",
               1, smallerMagnitude(-1, -2));
  assertNaN("Testing with illegal values",
            smallerMagnitude("bad", "values"));
  assertNaN("Testing with too few arguments",
            smallerMagnitude(1));
}
</script>
```


Test Page 1: Results

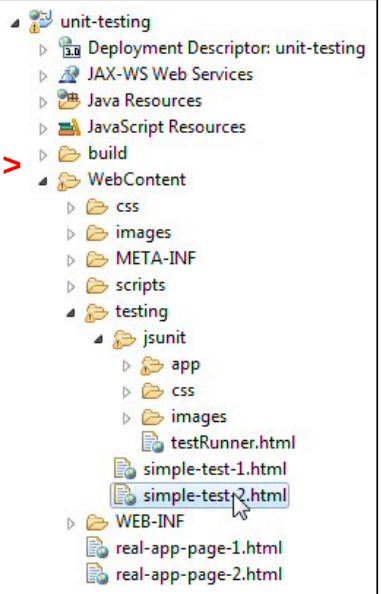
Deployed the app, then entered address of testRunner.html in browser address bar

Entered the URL of the test page (from previous slides) and pressed Run



Test Page 2: Part 1

```
<!DOCTYPE html>
<html>
<head><title>Unit Testing with JsUnit</title>
<script src="jsunit/app/jsUnitCore.js"></script>
</head>
<body>
```



Test Page 2: Part 2

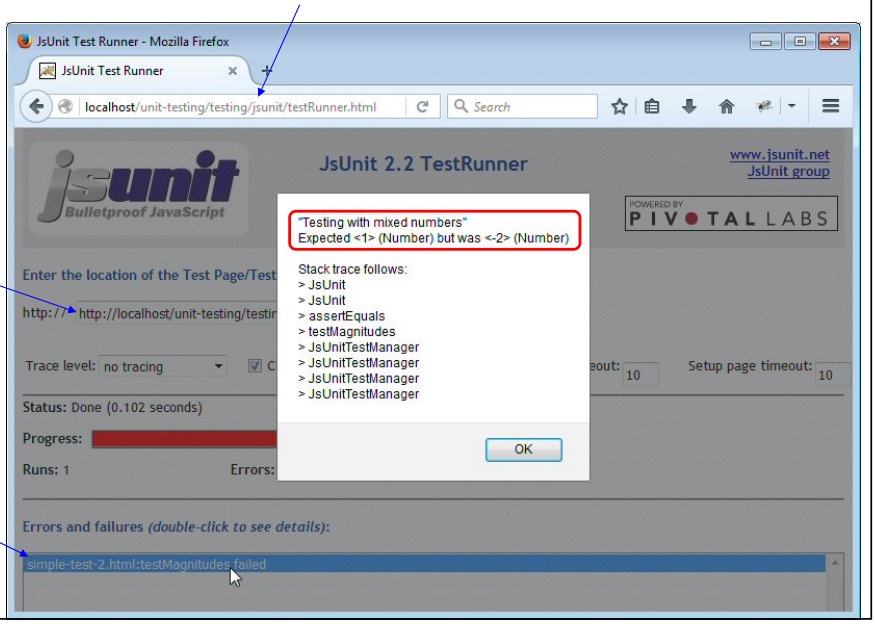
```
<script>
function smallerMagnitude(num1, num2) {
  return(Math.min(num1, num2)); // Forgot absolute value
}

function testMagnitudes() {
  assertEquals("Testing with positive numbers"
               1, smallerMagnitude(1, 2));
  assertEquals("Testing with mixed numbers",
               1, smallerMagnitude(-1, -2));
  assertNaN("Testing with illegal values",
            smallerMagnitude("bad", "values"));
  assertNaN("Testing with too few arguments",
            smallerMagnitude(1));
}
</script>
```

Test Page 2: Results

Deployed the app, then entered address of testRunner.html in browser address bar

Entered the URL of the test page (from previous slides) and pressed Run

Double-clicked on the error listing



More Realistic Example

Overview

- **Previous example**
  - Function to be tested (smallerMagnitude) was directly in the test page
    - Not realistic, since you want to test functions from your existing code
  - Function had no namespaces
    - Not realistic, since most real projects use namespaces to avoid name conflicts
- **New example**
  - Functions to be tested are part of JavaScript file used by existing application
  - Functions to be tested use namespaces
- **Project layout**
  - Uses testing subfolder with test pages that are distinct from main app pages
    - As before

JavaScript File: string-utils.js

```
var StringUtils = {};

// Returns the input string in reverse

StringUtils.reverse = function(string) {
  return(string.split("").reverse().join(""));
}

// Returns true if and only if the string reads the
// same way forward and backward (ignoring case differences).
// Checks a WORD, so does NOT strip out whitespace and punctuation.

StringUtils.isPalindrome = function(string) {
  string = string.toUpperCase();
  return(string == StringUtils.reverse(string));
}
```

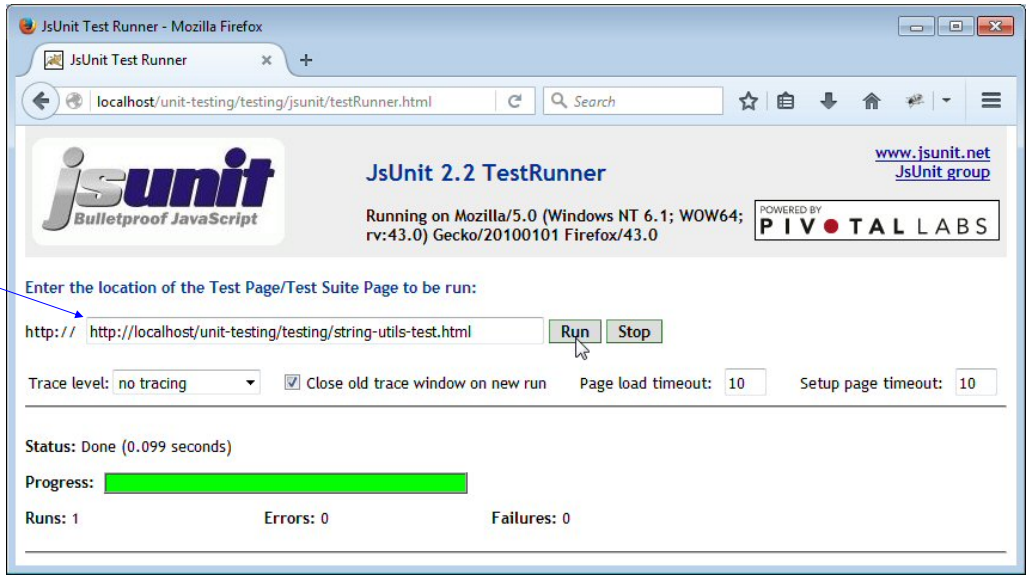
Test Page: Part 1

```
<!DOCTYPE html>
<html>
<head><title>Testing StringUtils functions</title>
<script src="../scripts/string-utils.js"></script>
<script src="jsunit/app/jsUnitCore.js"></script>
</head>
<body>
```


Test Page: Part 2

```
<script>
function testStrings() {
  assertEquals("Testing simple reverse",
               "halb", StringUtils.reverse("blah"));
  assertNotEquals("reverse should maintain cse",
                  "halb", StringUtils.reverse("Blah"));
  assertTrue("Testing simple palidrome",
             StringUtils.isPalindrome("bob"));
  assertTrue("isPalidrome should ignore case",
             StringUtils.isPalindrome("Bob"));
  assertTrue("Test of longer word (languge of Kerala)",
             StringUtils.isPalindrome("Malayalam"));
  assertFalse("isPalidrome should not ignore special characters",
              StringUtils.isPalindrome("Bob!"));
}
</script>
```

Test Page: Results



JavaScript File: math-utils.js

```
var MathUtils = {};

// Returns smallest magnitude (absolute value) of array of numbers.
//
// Special cases:
//  - Smallest of empty array should be Infinity.
//  - If array contains a non-number, result should be NaN.

MathUtils.smallestMagnitude = function(nums) {
  return(Math.min.apply(null, nums.map(Math.abs)));
}
```

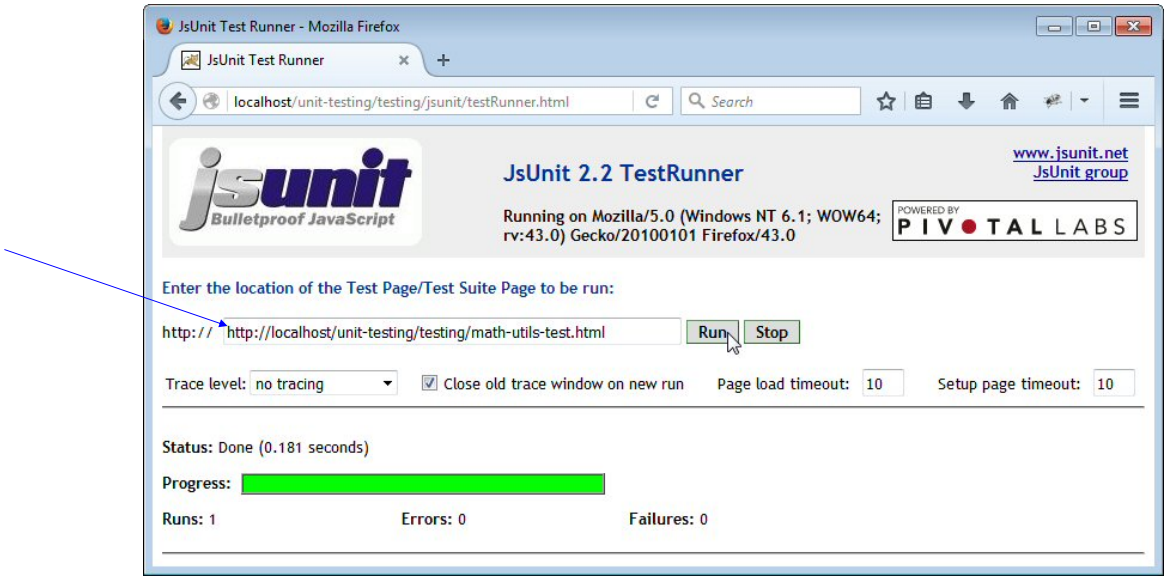
Test Page: Part 1

```
<!DOCTYPE html>
<html>
<head><title>Testing MathUtils functions</title>
<script src="../scripts/math-utils.js"></script>
<script src="jsunit/app/jsUnitCore.js"></script>
</head>
<body>
```


Test Page: Part 2

```
<script>
function testMagnitudes() {
  assertEquals("Smallest of positive nums is the min",
               1, MathUtils.smallestMagnitude([1,2,3]));
  assertEquals("Reordering array should make no difference",
               1, MathUtils.smallestMagnitude([3,1,2]));
  assertEquals("Absolute values should be used",
               1, MathUtils.smallestMagnitude([-3,-1,-2]));
  assertEquals("Smallest of empty array is Infinity",
               Infinity, MathUtils.smallestMagnitude([]));
  assertNaN("Non-numbers in array should result in NaN",
            MathUtils.smallestMagnitude(["foo", "bar"]));
}
</script>
```

Test Page: Results



Unit Testing with jQuery DOM Manipulation

Overview

- **Setup**
  - Load jquery.js
  - Load your-code-that-uses-jquery.js
  - Load jsUnitCore.js
- **Testing basic functions**
  - You will probably manipulate DOM before each assertion, then the assertions will look at DOM values
- **Testing Ajax**
  - Remember that Ajax calls are asynchronous
    - So, you cannot check for the results immediately after making the request
    - Instead, you must check in the response handler
  - Remember that the load function takes a response handler
    - To do extra side effects *after* main load has finished

JavaScript File: jquery-utils.js

```
var jQueryUtils = {};

// Clears the content of HTML elements

jQueryUtils.clear = function(selector) {
  $(selector).html("");
}

// Doubles (repeats) the content of HTML elements

jQueryUtils.double = function(selector) {
  var content = $(selector).html();
  $(selector).append(content);
}
```

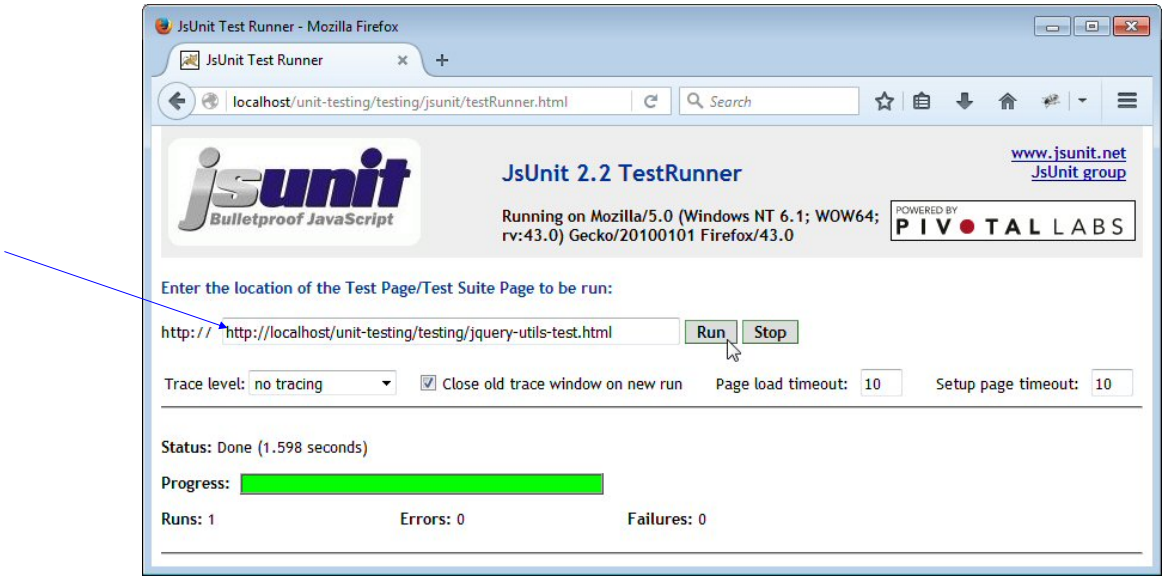
Test Page: Setup and HTML

```
<!DOCTYPE html>
<head><title>Testing jQueryUtils functions</title>
<script src="../scripts/jquery.js"></script>
<script src="../scripts/jquery-utils.js"></script>
<script src="jsunit/app/jsUnitCore.js"></script>
</head>
<body>
...
<div id="test-div-1">Blah</div>
<div id="test-div-2"></div>
```


Test Page: Testing DOM Manipulation

```
<script>
var region1 = "#test-div-1";

function testUtils() {
  assertEquals("Blah", $(region1).html());
  jQueryUtils.clear(region1);
  assertEquals("", $(region1).html());
  $(region1).html("Foo");
  jQueryUtils.double(region1);
  assertEquals("FooFoo", $(region1).html());
}

// Ajax tests shown in next section
</script>
```

Test Page: Results



Unit Testing with jQuery and Ajax

Overview

- **Idea: use setUpPage**
  - Invoke Ajax from function called setUpPage (not setupPage)
  - In the Ajax response handler, set the setUpPageStatus variable to "complete"
- **Why it works**
  - If this method exists, it is called before any of the tests
  - After calling setUpPage, JsUnit waits to call the tests until the value of the setUpPageStatus variable is "complete"
- **Not the same as more-widely-known setUp and tearDown**
  - Those methods are called before (setUp) and after (tearDown) each test function, and are used to reset variables and set up data. For example, if you have three test functions that all use the same data, it is less repetitive to set the data in setUp rather than repeat the setup code at the top of each test function.
  - The next test function runs immediately after setUp, and JsUnit does not wait for a certain variable to be set. So, these do not let you test Ajax results.

Need for setUpPage

- **You cannot check for results immediately after making request**
  - Because Ajax requests are asynchronous. So, the following fails because the assertEquals is likely to run before the result comes back from the server.

```
$(region).load("server-url");
assertEquals("server-text", $(region).html());
```

- **You cannot do assertions in the response handler**
  - Because JsUnit uses caught exceptions to handle assertions, and the Ajax response handler is not directly called by JsUnit. So, if assertion below fails, you see error in the JavaScript console, but you do *not* see red bar in JsUnit home page.

```
$(region).load("server-url",
               function(text) {
                 assertEquals("server-text", $(region).html());
               });
```

Test Page: Setup and HTML

```
<!DOCTYPE html>
<head><title>Testing jQueryUtils functions</title>
<script src="../scripts/jquery.js"></script>
<script src="../scripts/jquery-utils.js"></script>
<script src="jsunit/app/jsUnitCore.js"></script>
</head>
<body>
...
<div id="test-div-1">Blah</div>
<div id="test-div-2"></div>
```


Test Page: Testing Ajax

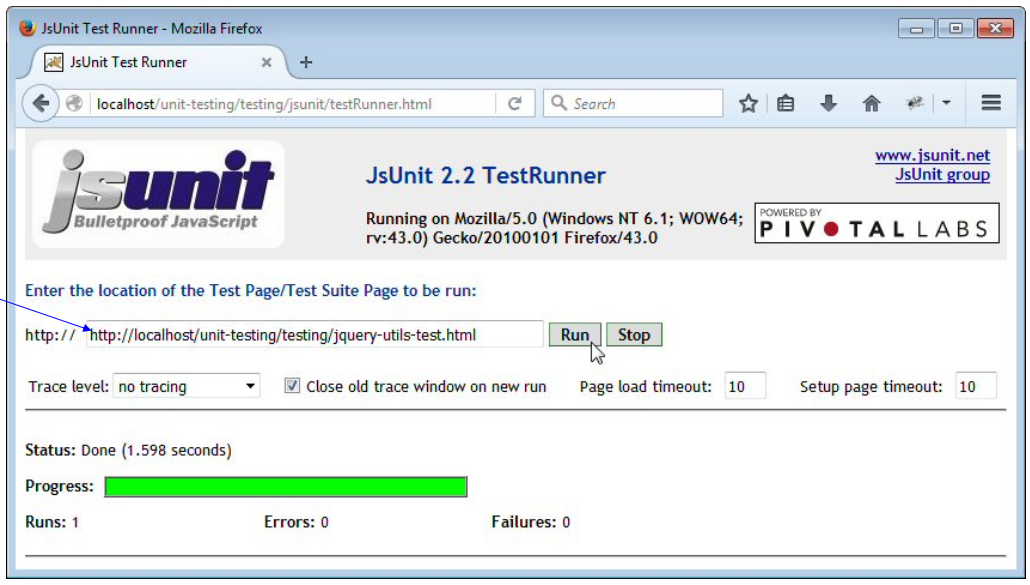
```
<script>
var region2 = "#test-div-2";

function setUpPage() {
  $(region2).load("server-data.txt",
                  function() { setUpPageStatus = "complete"; });
}

function testAjax() {
  assertEquals("Some Server String", $(region2).html());
}

// DOM manipulation tests show in previous section
</script>
```

After calling setUpPage, JsUnit waits, and does not call the test functions until this variable has the value "complete".

server-data.txt

```
Some Server String
```


Test Page: Results



Wrap-up

Summary

- **Download JsUnit**
  - http://sourceforge.net/projects/jsunit/files/
- **Copy the jsunit folder into your application**
  - your-app/testing/jsunit/...
- **Load jsunit/app/jsUnitCore.js in an HTML test page**
  - <script src="jsunit/app/jsUnitCore.js"></script>
- **In that page, make one or more functions called testBlah**
  - <script>function testMyCode() { ... }</script>
- **Use assertEquals, assertTrue, etc. in the testBlah function**
  - function testMyCode() { assertTrue(isPrime(13)); }
- **Load testRunner.html in a browser**
  - http://your-host/your-project/jsunit/testRunner.html
- **Enter address of your test page and click Run**
  - Enter http://your-host/your-project/your-test-page.html

coreservlets.com – custom onsite training 

Questions?

More info:
http://www.coreservlets.com/javascript-jquery-tutorial/ -- Tutorial on JavaScript, jQuery, and jQuery UI
http://courses.coreservlets.com/Course-Materials/java.html – General Java programming tutorial
http://www.coreservlets.com/java-8-tutorial/ – Java 8 tutorial
http://courses.coreservlets.com/java-training.html – Customized Java training courses, at public venues or onsite at *your* organization
http://coreservlets.com/ – JSF 2, PrimeFaces, Java 8, JavaScript, jQuery, Ext JS, Hadoop, RESTful Web Services, Android, HTML5, Spring, Hibernate, Servlets, JSP, GWT, and other Java EE training
Many additional free tutorials at coreservlets.com (JSF, Android, Ajax, Hadoop, and lots more)

For additional materials, please see http://www.coreservlets.com/. The JavaScript tutorial section contains complete source code for all examples in the entire tutorial series, plus exercises and exercise solutions for each topic.