



coding-bootcamps.com

Introduction to Blockchain Development
with Ethereum



Coding Bootcamps

By Jim Sullivan from [Coding Bootcamps](https://coding-bootcamps.com)

Testing Ethereum Contracts

About Instructor

- **Jim Sullivan** is a senior blockchain instructor and developer at [DC Web Makers](#).
- As a Software engineer with 18 years of experiences, Jim leads an expert team in Blockchain development, DevOps, Cloud, application development, and the SAFe Agile methodology.
- Jim is an expert in all blockchain platforms like Hyperledger, Ethereum, and Corda.

Prerequisite Courses

Taking the below courses are highly recommended:

[Intro to Blockchain Technology](#)

[Intro to JavaScript](#)

[Learn Node.JS, Express.JS and MongoDB](#)

Recap

- What we have learned so far?


Testing Ethereum Contracts

Ethereum Testing Frameworks

- Ethereum has testing frameworks for Remix and Truffle.
 - The approach is similar to nUnit testing, i.e. JUnit testing.
 - Tests can be written in Solidity or JavaScript.
 - The Ethereum contract testing is quite useful for:
 - Test Driven Development” (TDD)
 - Continuous Integration” (CI)
 - Regression Testing
 - Ethereum testing does not take the place of a strong quality strategy with an extensive test scripts to test all requirements.

Testing Ethereum Contracts

Remix Testing

- The Remix IDE has a testing module that will create contract frameworks for testing.
- The testing module can be access with the “Double Check” button. 
- Remix adds a “_test” suffix to the name of the test contract. Using the suffix, Remix recognizes the test contracts and bundles the tests into Remix’s Testing Module.

Testing Ethereum Contracts

Remix Testing

SOLIDITY UNIT TESTING

Test your smart contract by creating a foo_test.sol file (open ballot_test.sol to see the example). You will find more informations in the [documentation](#) Then use the stand alone NPM module remix-tests to run unit tests in your Continuous Integration <https://www.npmjs.com/package/remix-tests>. For more details, see How to test smart contracts guide in our documentation.

Generate test file

Run Tests ☒ Check/Uncheck all

☒ browser/ballot_test.sol

☒ browser/MyContract1_test.sol

Results:

browser/MyContract1_test.sol (test_1)

- ✓ (Check2)
- ✓ (Myfull int)
- ✓ (Check1)

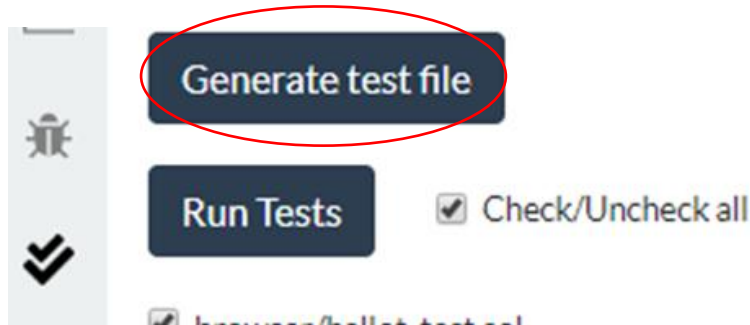
browser/MyContract1_test.sol

3 passing (2s)

Testing Ethereum Contracts

Remix Testing: Creating Tests

- Remix tests are created in the Remix testing module.
- When the **Generate test file** button in the Testing module is clicked, a testing file is generated with the “_test” suffix.
- The testing file only contains a framework to run contract tests.



Testing Ethereum Contracts

Remix Testing: Creating Tests

- When the testing file is generated, it only contains “out-of-the-box”, generic contract source code that makes a test framework.
- Some instructions on how to build tests are included in the test contract’s comments.
- No contract code or variables are brought into the test contract.
- The contracts and tests in the testing files must be edited to call the Smart Contract functions and variables.

Testing Ethereum Contracts

Remix Testing

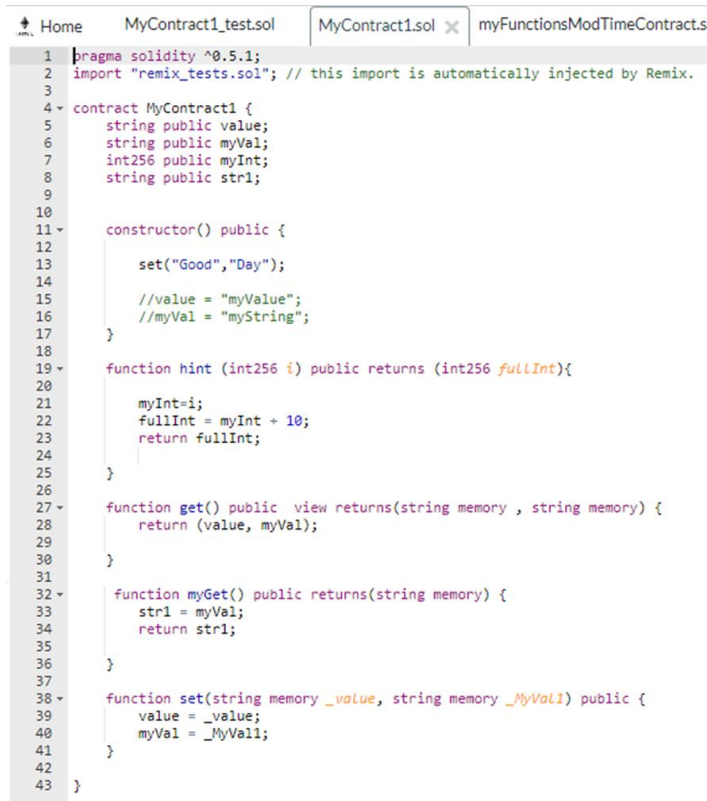
- The test contract is generated with contract Solidity code.
- The generic contract is used as a template.

```
1 pragma solidity >=0.4.0 <0.6.0;
2 import "remix_tests.sol"; // this import is automatically injected by Remix.
3
4 // file name has to end with '_test.sol'
5 contract test_1 {
6
7     function beforeAll() public {
8         // here should instantiate tested contract
9         Assert.equal(uint(4), uint(3), "error in before all function");
10    }
11
12    function check1() public {
13        // use 'Assert' to test the contract
14        Assert.equal(uint(2), uint(1), "error message");
15        Assert.equal(uint(2), uint(2), "error message");
16    }
17
18    function check2() public view returns (bool) {
19        // use the return value (true or false) to test the contract
20        return true;
21    }
22 }
23
24 contract test_2 {
25
26     function beforeAll() public {
27         // here should instantiate tested contract
28         Assert.equal(uint(4), uint(3), "error in before all function");
29    }
30
31    function check1() public {
32        // use 'Assert' to test the contract
33        Assert.equal(uint(2), uint(1), "error message");
34        Assert.equal(uint(2), uint(2), "error message");
35    }
36
37    function check2() public view returns (bool) {
38        // use the return value (true or false) to test the contract
39        return true;
40    }
41 }
```

Testing Ethereum Contracts

Remix Testing

- When a contract has its functions and variables tested using Remix testing, the contract will be instantiated in the test contract and the functions will be called in the test file.



```
1 pragma solidity ^0.5.1;
2 import "remix_tests.sol"; // this import is automatically injected by Remix.
3
4 contract MyContract1 {
5     string public value;
6     string public myVal;
7     int256 public myInt;
8     string public str1;
9
10
11     constructor() public {
12         set("Good","Day");
13
14         //value = "myValue";
15         //myVal = "myString";
16     }
17
18     function hint (int256 i) public returns (int256 fullInt){
19
20         myInt=i;
21         fullInt = myInt + 10;
22         return fullInt;
23     }
24
25     function get() public view returns(string memory , string memory) {
26         return (value, myVal);
27     }
28
29     function myGet() public returns(string memory) {
30         str1 = myVal;
31         return str1;
32     }
33
34     function set(string memory _value, string memory _MyVal1) public {
35         value = _value;
36         myVal = _MyVal1;
37     }
38 }
39
40
41
42
43 }
```

Testing Ethereum Contracts

Remix Testing

- The Remix testing file is built from the contract to be tested.
- The test contract (“_test”) instantiates the contract to be tested function and returns the results for validation.

The screenshot displays the Remix IDE interface. On the left, the 'SOLIDITY UNIT TESTING' panel shows instructions for creating a test file and a list of test files. The 'Run Tests' button is circled in red. Below it, the 'Results' section shows the test results for 'browser/MyContract1_test.sol (test_1)', with three green bars indicating successful tests: '✓ (Check2)', '✓ (MyfullInt)', and '✓ (Check1)'. The right panel shows the Solidity code for 'MyContract1_test.sol', which includes a pragma statement, imports, and test functions like 'beforeAll', 'check1', and 'check2'.

SOLIDITY UNIT TESTING

Test your smart contract by creating a foo_test.sol file (open ballot_test.sol to see the example). You will find more informations in the [documentation](https://www.npmjs.com/package/remix-tests) Then use the stand alone NPM module remix-tests to run unit tests in your Continuous Integration For more details, see [How to test smart contracts guide](https://www.npmjs.com/package/remix-tests) in our documentation.

Generate test file

Run Tests ☐ Check/Uncheck all

- ☐ browser/ballot_test.sol
- ☒ browser/MyContract1_test.sol
- ☐ browser/test_test.sol

Results:

browser/MyContract1_test.sol (test_1)

- ✓ (Check2)
- ✓ (MyfullInt)
- ✓ (Check1)

browser/MyContract1_test.sol
Passing (2s)

MyContract1_test.sol

```
1 pragma solidity ^0.5.1;
2 import "remix_tests.sol"; // this import is automatically injected by Remix.
3 import "../MyContract1.sol";
4
5 // file name has to end with '_test.sol'
6 contract test_1 {
7
8     int256 public myfullInt = 0;
9     string myValue1;
10    string theValue;
11    MyContract1 mycontract1;
12    string strVal;
13
14
15    function beforeAll() public {
16
17        mycontract1 = new MyContract1();
18        // here should instantiate tested contract
19        //Assert.equal(uint(4), uint(3), "error in before all function");
20    }
21
22    function check1() public {
23        // use 'Assert' to test the contract
24
25        myfullInt = mycontract1.hint(5);
26        mycontract1.set("Hello", "World");
27
28        Assert.equal(myfullInt, 15, "Not the Correct sum");
29
30        strVal = mycontract1.myGet();
31
32        Assert.equal(mycontract1.value(), "Hello", "Error: Not Hello");
33        Assert.equal(strVal, "World", "Error: not World");
34    }
35
36
37
38
39    function check2() public view returns(string memory, string memory) {
40
41        mycontract1.get();
42    }
43
44
45
46
47 }
```

Testing Ethereum Contracts

The Remix tests include a group of functions to validate contracts.

These functions evaluate returned values and run equality or inequality comparisons with expected values. These functions are called the Assert functions.

Remix Testing functions	
Function	Parameter
Assert.ok()	bool
Assert.equal()	uint, int, bool, address, bytes32, string
Assert.notEqual()	uint, int, bool, address, bytes32, string
Assert.greaterThan()	uint, int
Assert.lessThan()	uint, int

Testing Ethereum Contracts

Remix Testing

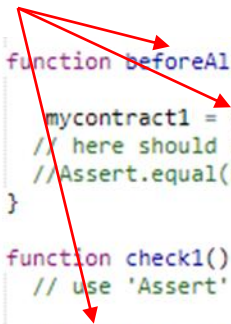
- The Assert functions follow a template for evaluations.
- The template follows: `Assert.equal(returned value, expected value, "message")`
- Available special functions for Solidity Testing:
 - `beforeEach()` - runs before each test to create new instance of the contract for testing.
 - `beforeAll()` – instantiates the contract to be tested
 - `before()` function runs before tests, instantiates the contract to be tested, and initializes.

Testing Ethereum Contracts

Remix Testing

```
15  function beforeAll() public {  
16  
17      mycontract1 = new MyContract1();  
18      // here should instantiate tested contract  
19      //Assert.equal(uint(4), uint(3), "error in before all function");  
20  }  
21
```

Calling the instantiated contracts, functions and variables.



```
15  function beforeAll() public {  
16  
17      mycontract1 = new MyContract1();  
18      // here should instantiate tested contract  
19      //Assert.equal(uint(4), uint(3), "error in before all function");  
20  }  
21  
22  function check1() public {  
23      // use 'Assert' to test the contract  
24  
25      myfullInt = mycontract1.hint(5);  
26      mycontract1.set("Hello","World");  
27
```


Testing Ethereum Contracts

Node Package Manager and remix-tests

- The NPM package for Remix tests is called: remix-tests.
- Docs: <https://www.npmjs.com/package/remix-tests>
- The package can be installed using:
`npm -g install remix-tests.`

Testing Ethereum Contracts

Truffle Testing with JavaScript

- Truffle includes an automated testing framework.
- The framework supports simple and manageable tests in two different ways:
 - JavaScript
 - Solidity
- JavaScript tests use the Truffle console to instantiate contracts, call functions and evaluates results.
- JavaScript tests use the `async()/await()` functions demonstrated earlier.

Testing Ethereum Contracts

Truffle Testing with JavaScript

- The truffle JavaScript tests also have initialization function such as the, `beforeEach` function.
- The truffle JavaScript tests use their own `Assert` functions to evaluate results.
- The JavaScript tests have the functions:
 - `require()`
 - `before()`
 - `async()`
 - `await()`
 - `assert()`, for equality, or inequality

Testing Ethereum Contracts

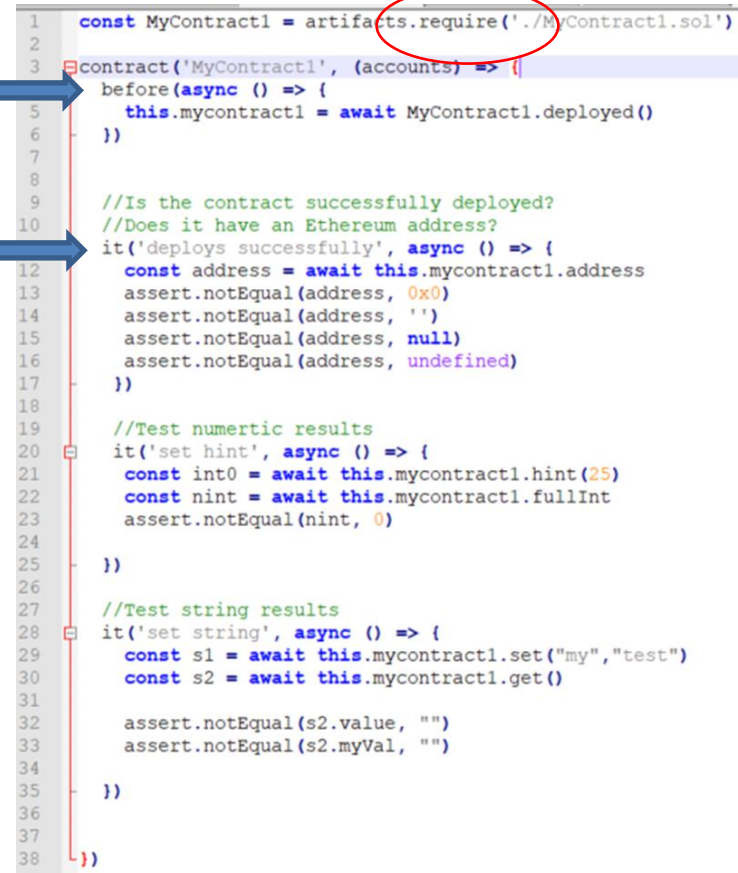
Truffle Testing with JavaScript

- Test runs in an **it()** function block.
- The **it()** function includes the test name and lists of contract functions and result evaluation.
- JavaScript testing uses the **assert()** functions for evaluations of equality and inequality, and/or pass/fail.

Testing Ethereum Contracts

Truffle Testing with JavaScript

- The JavaScript test validates deployment, and it calls functions that return integers and strings



```
1  const MyContract1 = artifacts.require('./MyContract1.sol')
2
3  contract('MyContract1', (accounts) => {
4    before(async () => {
5      this.mycontract1 = await MyContract1.deployed()
6    })
7
8
9    //Is the contract successfully deployed?
10   //Does it have an Ethereum address?
11   it('deploys successfully', async () => {
12     const address = await this.mycontract1.address
13     assert.notEqual(address, 0x0)
14     assert.notEqual(address, '')
15     assert.notEqual(address, null)
16     assert.notEqual(address, undefined)
17   })
18
19   //Test numeric results
20   it('set hint', async () => {
21     const int0 = await this.mycontract1.hint(25)
22     const nint = await this.mycontract1.fullInt
23     assert.notEqual(nint, 0)
24
25   })
26
27   //Test string results
28   it('set string', async () => {
29     const s1 = await this.mycontract1.set("my", "test")
30     const s2 = await this.mycontract1.get()
31
32     assert.notEqual(s2.value, "")
33     assert.notEqual(s2.myVal, "")
34
35   })
36
37 })
38
```

Testing Ethereum Contracts

Truffle Testing with JavaScript

- The JavaScript tests run using command: `truffle test`
- The 3 JavaScript tests run successfully.



Command Prompt

```
C:\ethereumTesting>truffle test
Using network 'development'.
```

```
Contract: MyContract1
  ✓ deploys successfully
  ✓ set hint (77ms)
  ✓ set string (173ms)
```

```
3 passing (290ms)
```

```
C:\ethereumTesting>
```

Ethereum Client-Side Applications

Summary

- Ethereum testing supports test automation, Test development, and continuous integration
- Remix testing module
- The Building Remix tests
- Running Remix tests
- Truffle tests with JavaScript
- Building JavaScript tests: require, before, await, async, asset
- it() Functions
- Running JavaScript tests with Truffle

Assessment

1. Ethereum Contract testing support Continuous Integration and ...?
2. True or False: Remix supports test building?
3. True or False: Remix test module only generates a testing template?
4. True or False: Remix testing module organizes tests?
5. True or False: Remix can run contract tests?
6. True or False: Remix tests cannot be run outside of Remix?
7. True or False: JavaScript contract tests do not support automation, test driven development?
8. True or False: JavaScript testing can validate if a deployed contract has an Ethereum address.
9. True or False: JavaScript tests have functions for evaluation
10. True or False: JavaScript tests do not return real time results

More Resources

- [History and Evolution of Blockchain Technology from Bitcoin](#)
- [Overview of Blockchain evolution and phases from Ethereum to Hyperledger](#)
- [Comprehensive overview and analysis of blockchain use cases in many industries](#)
- [Overview of blockchain technology and blockchain development](#)

More Resources...

- [How to Write Ethereum Smart Contracts with Solidity in 1 hour](#)
- [How to Build Auction DApp With Ethereum and Solidity Programming Language](#)
- [How to Work with Ethereum Blockchain Applications through Remix IDE](#)
- [Certified Solidity Professional Certification](#) exam

More Blockchain Training

- [Blockchain Management with Hyperledger for System Admins](#)
- [Hyperledger Fabric and Composer for Developers](#)
- [Intro to Blockchain Cybersecurity](#)
- [Learn Solidity Programming by Examples](#)
- [Learn Blockchain Dev with Corda R3](#)
- [Intro to Hyperledger Sawtooth for System Admins](#)

Next Session

Ethereum Use Cases



coding-bootcamps.com

Thank you



Coding
Bootcamps