



coding-bootcamps.com

Introduction to Blockchain Development
with Ethereum



Coding Bootcamps

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

Smart Contracts Functions

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?

Smart Contracts Functions

Functions

- In Ethereum, a Function is executable block that performs an operation.
- Functions can take parameters and/or return parameters.
- Functions need to be specified as public, private, internal, or external.
- Public functions are visible and they can be called by other function or by the contract interface.

```
23 ▾ function get() public view returns(string memory, string memory) {  
24     return (value, myVal);  
25 }  
26  
27  
28 ▾ function set(string memory _value, string memory _MyVal1) public {  
29     value = _value;  
30     myVal = _MyVal1;  
31 }  
32 }
```

Smart Contracts Functions

Functions

- Private functions are only visible within their own contract.
- Private functions can only be called by other functions within their own contract and cannot be called by other contracts.
- Functions declared as external can be called from other contracts.
- External functions cannot be called internally.

```
16 ▾  
17  
18  
19  
20  
21  
--  
  
function setint (int256 i) external {  
  
    myInt=i;  
    fullInt = myInt + 10;  
  
}
```

Smart Contracts Functions

```
16 ▾ function setint (int256 i) private {  
17  
18     myInt=i;  
19     fullInt = myInt + 10;  
20  
21 }  
22  
23 ▾ function set(string memory _value, string memory _MyVal1) public {  
24     value = _value;  
25     myVal = _MyVal1;  
26     setint(j);  
27 }  
28  
29 ▾ function get() public view returns(string memory, string memory, int256 _fullint) {  
30     return (value, myVal, fullInt);  
31  
32 }  
33 }
```


Smart Contracts Functions

Functions: Calling functions from other Contracts

- Functions and data can be called from other contracts.
- Contracts can be declared within other contracts using the **new** command.
- Once a contract is declared, within another contract, the newly declared contract's public functions and data can be called.

Smart Contracts Functions

```
1  pragma solidity >=0.5.1;
2
3  contract holdDaata {
4      uint public data = 42;
5      //hd.myBaseData(20);
6
7      function setMyData (uint256 _i) public pure returns (uint256 _prd) {
8          //Doubles the past in data
9          _prd = _i * 2;
10     }
11
12     function myBaseData(int256 _m) internal pure returns (int256 _r){
13
14         _r = _m + 10;
15     }
16 }
17
18
19 contract userBase {
20
21     holdDaata hd = new holdDaata();
22     uint256 theData;
23
24     function lookAtData() public view returns (uint) {
25         return hd.data();
26     }
27
28     function getMyData(uint256 _iData) public view returns (uint256 _dataval){
29
30         _dataval = hd.setMyData(_iData);
31
32         return _dataval;
33     }
34 }
35 }
```

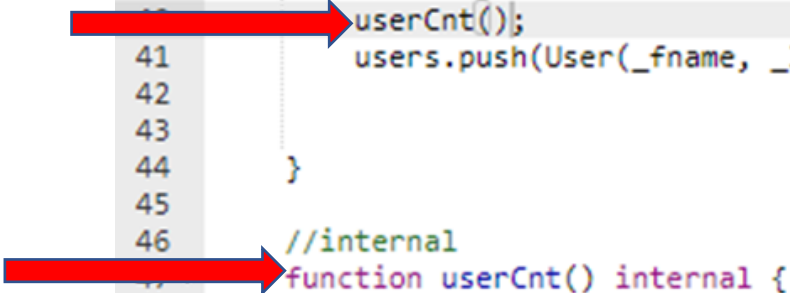
Smart Contracts Functions

Functions: Calling Functions

- Solidity supports calling functions from other functions.
- A function call may or may not return data
- A function call may or may not take input parameters.
- A function call is performed the same way it is performed in Java, C++, or JavaScript.

Smart Contracts Functions

```
37  function addUser(string memory _fname, string memory _lname) public onlyOwner {  
38  
39      //adding internal function  
40      userCnt();  
41      users.push(User(_fname, _lname));  
42  
43  
44  }  
45  
46  //internal  
47  function userCnt() internal {  
48      userCounter += 1;  
49  
50  }  
51
```

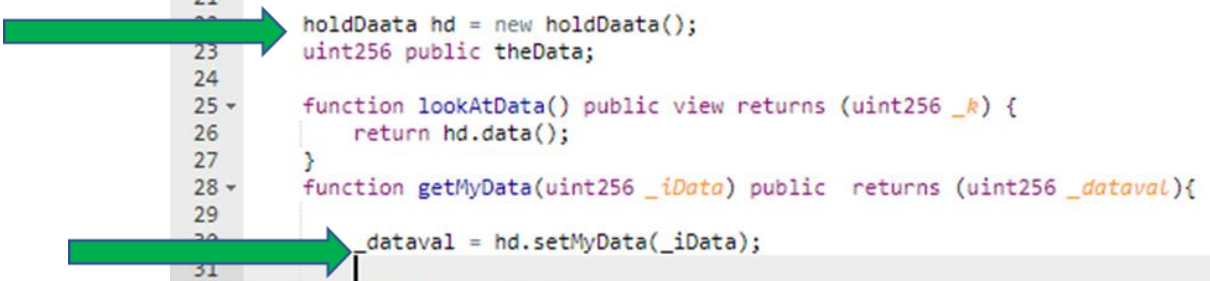


Smart Contracts Functions

Functions: Calling Functions

- As discussed earlier, functions can be called from other contracts.
- The other contract must be instantiated using **new**.

```
20 contract userBase {  
21  
22     holdDaata hd = new holdDaata();  
23     uint256 public theData;  
24  
25     function lookAtData() public view returns (uint256 _k) {  
26         return hd.data();  
27     }  
28     function getMyData(uint256 _iData) public returns (uint256 _dataval){  
29  
30         _dataval = hd.setMyData(_iData);  
31  
32         theData = _dataval;  
33         return theData;  
34     }  
35 }  
36 }
```



Smart Contracts Functions

Functions: Inheritance

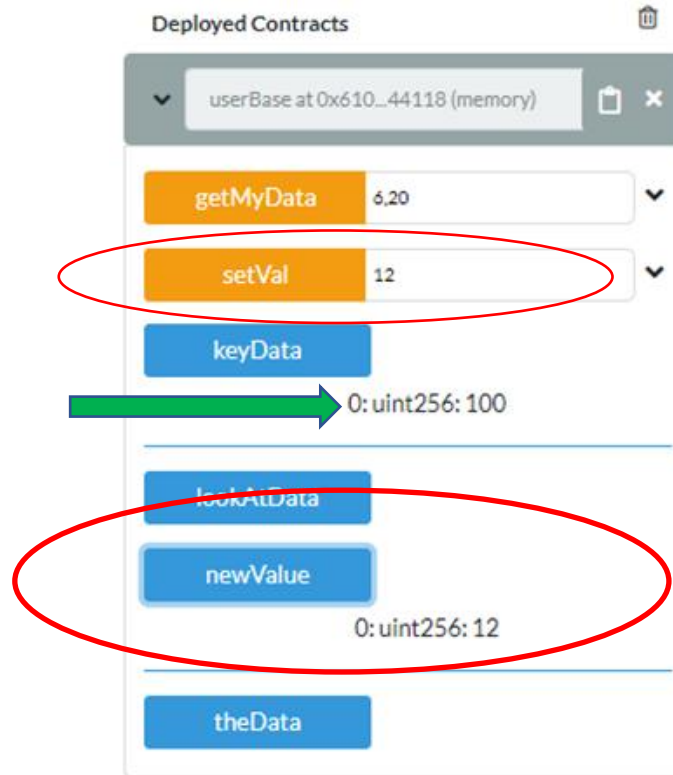
- Solidity supports inheritance and polymorphism similar to Java and C++
- Inheritance between contract is implemented using **is** keyword
- Inheritance is defined in the function header, and multiple inheritance is supported.

Smart Contracts Functions

```
20 contract basicData {  
21  
22     uint256 public keyData = 100;  
23     uint256 public newValue;  
24  
25     function setVal(uint256 _iVal) public {  
26         newValue = _iVal;  
27     }  
28 }  
29  
30 contract userBase is basicData {  
31  
32     holdDaata hd = new holdDaata();  
33     uint256 public theData;  
34  
35     function lookAtData() public view returns (uint256 _k) {  
36         return hd.data();  
37     }  
38  
39     function getMyData(uint256 _iData, uint256 _imyData) public returns (uint256 _dataval, uint256 _key){  
40  
41         _dataval = hd.setMyData(_iData);  
42         setVal(_imyData);  
43         theData = _dataval;  
44         return (theData, keyData);  
45     }  
46 }  
47  
48
```



Smart Contracts Functions



Smart Contracts Functions

Functions: Returns and Return

- When a function returns data, it is declared in the header, and in the function's statement.
- If the function statement returns the data declared in the header, return is not needed. No return statement is needed.

```
contract Simple {  
    function arithmetic(uint _a, uint _b)  
        public  
        pure  
        returns (uint o_sum, uint o_product)  
    {  
        o_sum = _a + _b;  
        o_product = _a * _b;  
    }  
}
```

```
contract Simple {  
    function arithmetic(uint _a, uint _b)  
        public  
        pure  
        returns (uint o_sum, uint o_product)  
    {  
        return (_a + _b, _a * _b);  
    }  
}
```

Smart Contracts Functions

Functions: pure

- Pure functions do not update or read the transaction or block state values.
- Because hashes are sensitive to changes, determinism is key in Blockchain.
- Deterministic functions return the same value every time they are run.
- Pure functions do access state values such as:
address.balance
- Pure functions do not access state variables: block, tx, msg
- Pure functions only call other pure functions.

Smart Contracts Functions

```
1 pragma solidity >=0.5.0 <0.7.0;
2
3 contract addEm {
4     function myAdd(uint256 numVal, uint256 baseVal) public pure returns (uint _a) {
5
6         _a = numVal * baseVal + 16;
7         testP("test Pure");
8     }
9
10
11     function testP(string memory _mystr) public returns (string memory _str){
12
13         _str = _mystr;
14     }
15 }
16 }
```

```
1 pragma solidity >=0.5.0 <0.7.0;
2
3 contract addEm {
4     function myAdd(uint256 numVal, uint256 baseVal) public pure returns (uint _a) {
5
6         _a = numVal * baseVal + 16;
7         testP("test Pure");
8     }
9
10
11     function testP(string memory _mystr) public pure returns (string memory _str){
12
13         _str = _mystr;
14     }
15 }
16 }
```

Smart Contracts Functions

Functions: view

- Just like pure functions, view functions do not modify the state.
- View functions do not write to state variables or emit events.
- View functions do not create other contracts or make Ether calls.
- View functions only call other functions declare as view or pure.

```
1  pragma solidity >=0.5.0 <0.7.0;
2
3  contract addEm {
4      function myAdd(uint256 numVal, uint256 baseVal) public view returns (uint _a) {
5
6          _a = numVal * baseVal + 16 + now;
7
8      }
9
10 }
```

Smart Contracts Functions

Functions: Function Modifiers

- Function Modifiers, as the name implies, change the way a function executes.
- Modifier are like custom function types.
- Modifiers are called in the function's header and can test conditions during run time.
- Modifiers are inheritable.

Smart Contracts Functions

```
46 ▾ modifier onlyadminUser() {  
47  
48     require (msg.sender == admin);  
49     _;  
50  
51 }  
52  
53  
54 ▾ function addUser(string memory _fname, string memory _lname) public onlyadminUser {  
55  
56     //adding internal function  
57     addUserCnt();  
58     users[usrCounter] = User(usrCounter, _fname, _lname);  
59 }  
60  
61
```

Smart Contracts Functions

Functions: Constructors

- Constructors are optional functions used for contract initialization.
- Constructors are the first to execute when the contract runs.
- Once the constructor runs, the whole contract is deployed to the blockchain.
- Constructors are either public or private.
- If the constructor is omitted, a default Solidity runs a default constructor.

Smart Contracts Functions

```
1 pragma solidity ^0.5.1;
2
3 contract myAddress {
4
5     //mapping is a key / value pair
6     mapping(address => uint256 ) public balance;
7     address payable mywallet1;
8
9     event Purchase (
10         address _buyer,
11         uint256 _amount
12     );
13 };
14 //event are for subscribing and filtering for an event.
15
16 constructor (address payable _wallet) public {
17     mywallet1 = _wallet;
18 }
19
20
21 function() external payable {
22
23     buyAsset();
24 }
25
26
27 function buyAsset() public payable{
28     //buy
29
30     balance[msg.sender] += 1;
31     mywallet1.transfer(msg.value);
32     //send ether to a wallet
33     emit Purchase(msg.sender, 1);
34 }
35
36
37
38 }
```


Smart Contracts Functions

Functions: Overloading Functions

- Ethereum supports overloading functions
- A contract can have more than one function with the same name as long as parameters are different.
- At a runtime, the contract runs the called function whose parameters match.

Smart Contracts Functions



```
60 ▾ function addUser(string memory _fname, string memory _lname) public {  
61  
62     //adding internal function  
63     addUser();  
64     users[userCounter] = User(userCounter, _fname, _lname);  
65 }  
66  
67 ▾ function addUser() internal {  
68     userCounter += 1;  
69 }  
70  
71
```



addUser

"Mike", "Sims"



Smart Contracts Functions

Functions: Ethereum Time

- The Ethereum Blockchain uses the epoch time.
- It is a common, maintained time from a specific point in time called the epoch.
- Epoch time is the number of seconds since the epoch.
- The epoch can be converted to a date and time.
- Function modifiers can read Ethereum time.
- The current time can be return by using `block.timestamp`.
- The current time can be compared to an epoch time in the future.

Smart Contracts Functions



EpochConverter

Epoch & Unix Timestamp Conversion Tools

The current Unix epoch time is 1567968904

```
uint256 openingTime = 1567612315 ;
```

```
modifier onlyWhileOpen() {
```

```
    require (block.timestamp >= openingTime);
```

```
    _;
```

```
}
```

```
function addUser(string memory _fname, string memory _lname) public onlyWhileOpen {
```

```
    //adding internal function
```

```
    addUser();
```

```
    users[usrCounter] = User(usrCounter, _fname, _lname);
```

```
}
```

Smart Contracts Functions

Functions: address payable

- Address payable is an Ethereum/ Solidity data type that can receive Ether.
- Address Payable is used when transaction results in payments made to other Ethereum addresses

```
1  pragma solidity ^0.5.1;
2
3  contract mySendReceiveEther {
4
5      //mapping is a key / value pair
6      mapping(address => uint256 ) public balance;
7      address payable mywallet1;
8
9      event Purchase (
10         address _buyer,
11         uint256 _amount
12     );
13 }
```

Structure of a Smart Contract

Summary

- The Smart Contract function: passing and returning data
- The Smart Contract function visibility: public, private, etc
- Calling and declaring functions
- Inheritance
- Pure and view functions
- Function modifiers: time, owner
- Constructors
- Overloading Functions
- Payable functions

Assessment

1. Is a function a block of contract code?
2. True or False: Contract function cannot take inputs?
3. True or False: All functions are private?
4. True or False: A function inherit from another functions
5. True or False: Pure function can run differently each time?
6. True or False: View functions cannot call other View functions?
7. True or False: Constructors initialize contracts?
8. True or False: Overloaded functions are evaluated by parameters?
9. True or False: Function modifiers can evaluate conditions.
10. True or False: Function can make payments

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 Client-Side Applications



coding-bootcamps.com

Thank you



Coding
Bootcamps