



coding-bootcamps.com

Introduction to Blockchain Development
with Ethereum



Coding Bootcamps

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

Structure of a Smart Contract

Smart Contract Constructs

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?

Structure of a Smart Contract

Ethereum Development

- Ethereum Smart Contracts can be written in Solidity, Vyper, JavaScript or other High Level programming languages.
- However, no matter the programming language, they all compile to the EVM, and are deployed to the Ethereum networks.
- Smart Contracts are meant to respond to events and take actions on transactions.
- Smart Contracts can manage Ether, send and receive Ether for payments.

Structure of a Smart Contract

Ethereum Development: Pragma

- The pragma key word is used to invoke the certain compilers.
- Solidity contract writers will be different variations of pragma entries.
- The pragma directive is entered as shown:
- Using a “greater than or equal to compiler” indicator: `pragma solidity >=0.4.22 <0.6.0;`
- The contract with the entry below, with the caret, ^, will not compile with a version earlier than 0.5.2
- `pragma solidity ^0.5.2;`
- Other variations: `pragma solidity >=0.5.0 <0.7.0;`

Structure of a Smart Contract

Ethereum Development: Comments

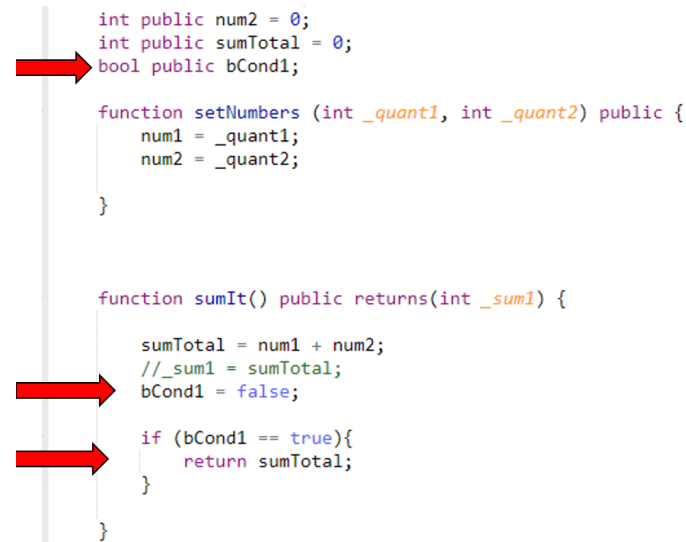
- Code comments greatly increase the maintainability and understandability of your contract.
- Single line comments are indicated by “//”
- Multiline comments are entered as : “/* */”

```
1 pragma solidity >=0.5.0 <0.7.0; |
2
3 contract TotalUpContract {
4
5     // This contract totals passed in numbers
6
7     /* Declare variable num1
8      Declare variable num2
9      Declare sumToal
10     initialize all to 0 */
11
12     int public num1 = 0;
13     int public num2 = 0;
14     int public sumTotal = 0;
```


Structure of a Smart Contract

Ethereum Development: Data Types

- Booleans are either “true” or “false”
- Booleans have the list of operators
 - ! negation
 - && Logical conjunction, “and”
 - || Logical disjunction, “or”
 - == equality
 - != inequality



```
int public num2 = 0;
int public sumTotal = 0;
bool public bCond1;

function setNumbers (int _quant1, int _quant2) public {
    num1 = _quant1;
    num2 = _quant2;
}

function sumIt() public returns(int _sum1) {
    sumTotal = num1 + num2;
    // _sum1 = sumTotal;
    bCond1 = false;

    if (bCond1 == true){
        return sumTotal;
    }
}
```

The diagram illustrates the structure of a smart contract with three red arrows pointing to specific lines of code. The first arrow points to the declaration of the `bool public bCond1` variable. The second arrow points to the assignment of `bCond1 = false` within the `sumIt` function. The third arrow points to the `if (bCond1 == true)` conditional statement within the same function.

Structure of a Smart Contract

Ethereum Development: Data Types

- Solidity supports signed and unsigned integers
 - int (signed)
 - uint (unsigned)
 - int8
 - int256
 - uint
 - uint256

Structure of a Smart Contract

Ethereum Development: Data Types

- The integer operators exhibit the same functionality and behavior as Java, C++ and JavaScript.
 - +, addition
 - -, subtraction
 - *, multiplication
 - /, division
 - %, modulo
 - **, exponential

Structure of a Smart Contract

The integer operators exhibit the same functionality as Java, C++ and JavaScript.

- +, addition
- -, subtraction
- *, multiplication
- /, division
- %, modulo
- **, exponential
- <=, less than or equal to
- <, less than
- ==, equality
- !=, inequality
- >=, greater than or equal to
- >, greater than

Structure of a Smart Contract

Ethereum Development: Strings

- Strings are special type of character arrays.
- Strings must be inside quotes or double quotes: “hello “, ‘world ‘
- Strings cannot be directly compared for equivalence in Solidity
- However, strings’ hash values can be compared.

```
function sumIt() public returns(int _sum1, string memory _str1) {  
    sumTotal = num1 + num2;  
    bCond1 = true;  
    _str1 = instruct;  
    bEqual = compareStringsbyBytes(_str1, "test");  
  
    if (bCond1 == true && bEqual == true){  
        return (sumTotal, instruct);  
    }  
}  
  
function compareStringsbyBytes(string memory s1, string memory s2) internal pure returns(bool){  
    return keccak256(abi.encodePacked(s1)) == keccak256(abi.encodePacked(s2));  
}
```

Structure of a Smart Contract

Ethereum Development: address data type

- The address data type is meant for Ethereum addresses.
- There are 2 versions of the address data type: address and address payable.
- address payable: same as address, but also including methods: transfer and send.
- address payable supports sending Ether (Ethereum's native currency) to another address.
- address uses the same comparisons as integer.

Structure of a Smart Contract

Ethereum Development: address data type

```
1  pragma solidity 0.5.1;
2
3  contract GetAddress {
4
5      address payable public wallet1;
6
7      function getMyAddress() public returns (address _addr1){
8
9          wallet1 = msg.sender;
10         return wallet1;
11     }
12
13
14
15 }
```

Structure of a Smart Contract

Ethereum Development: Enums

- Enums are an example of a user defined or custom data type.
- Enums are given a specific name and contain a list of data.
- The data is 0 indexed.
- When an enum is declared, it is declared with its name and it is typed as the enum.

```
3 contract myStateContract {  
4     enum UserOptions {listen, connect, cancel}  
5     UserOptions public myOption;  
6  
7     constructor() public {  
8         myOption = UserOptions.cancel;  
9     }  
10  
11     function connect() public {  
12         myOption = UserOptions.connect;  
13     }  
14 }
```


Structure of a Smart Contract

Ethereum Development: Structs

- Another custom data type is Structs
- Structs contain a list of attribute datatypes that describe it.
struct Funder {
 address addr;
 uint amount;
}
- Structs model real-world items that can be managed with a contract. Structs are data structures.
- Structs are used by other data structures such as Arrays and Mappings.

Structure of a Smart Contract

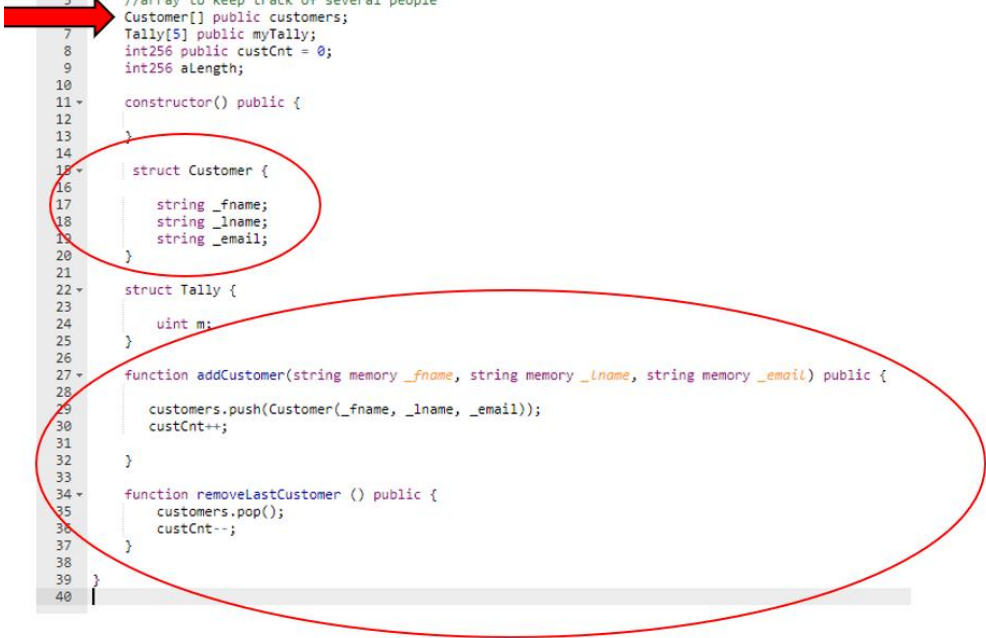
Ethereum Development: Arrays

- Arrays are data structures where one variable of a certain data type, contains multiple values, in different sections of memory.
- Arrays can be fixed size or dynamic.
- Fixed size arrays can be declared at the compile time.
- Values can also be added or removed from the Array using member methods push and pop.
- The array length is used to add length to the array.

Structure of a Smart Contract

Ethereum Development: Arrays

```
1  pragma solidity 0.5.1;  
2  
3  contract ArrayContract {  
4  
5      //array to keep track of several people  
6      Customer[] public customers;  
7      Tally[5] public myTally;  
8      int256 public custCnt = 0;  
9      int256 aLength;  
10  
11     constructor() public {  
12  
13     }  
14  
15     struct Customer {  
16  
17         string _fname;  
18         string _lname;  
19         string _email;  
20     }  
21  
22     struct Tally {  
23  
24         uint m;  
25     }  
26  
27     function addCustomer(string memory _fname, string memory _lname, string memory _email) public {  
28  
29         customers.push(Customer(_fname, _lname, _email));  
30         custCnt++;  
31     }  
32  
33     function removeLastCustomer () public {  
34  
35         customers.pop();  
36         custCnt--;  
37     }  
38  
39 }  
40
```



Structure of a Smart Contract

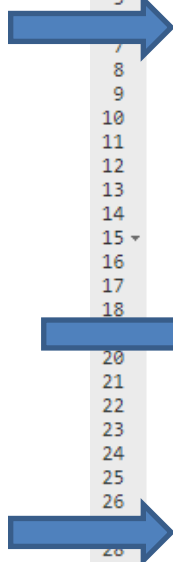
Ethereum Development: Mappings

- Mapping are like hash tables in that data can be accessed with a key.
- Mappings are data structures stored with key, value pairs.
- Syntax: `mapping(key => value)`
- Example:
`customers(1,"Mark", "Smith")`

Structure of a Smart Contract

Ethereum Development: Mappings

```
3 contract MyPayVendorContract {
4
5     //mapping to keep track of several vendors
6     mapping(uint => Vendor) public vendors;
7     mapping (address => uint) public balance;
8
9     uint256 public vendCounter = 0;
10    address payable public issuer;
11    uint public issuerBal;
12
13    event Sent(address payable from, address to, string name, uint amount);
14
15    constructor() public {
16        issuer = msg.sender;
17    }
18
19    struct Vendor {
20
21        uint256 _cid;
22        string _name;
23        address payable _addr;
24        uint256 _owedAmount;
25    }
26
27    function addVendor(string memory _name, address payable _addr, uint256 _owedAmount) public {
28
29        vendors[vendCounter] = Vendor(vendCounter, _name, _addr, _owedAmount);
30        vendCounter++;
31    }
32
33    //continue
```



The diagram illustrates the structure of a smart contract with three blue arrows pointing to specific parts of the code:

- The first arrow points to the `mapping(uint => Vendor) public vendors;` line (line 6), which defines a mapping to keep track of several vendors.
- The second arrow points to the `struct Vendor {` line (line 19), which defines the structure of a vendor.
- The third arrow points to the `function addVendor` line (line 27), which defines the function to add a new vendor.

Structure of a Smart Contract

Ethereum Development: event and emit

- Events are members of contracts
- When an event is called, its arguments are stored in the transaction's log
- The logged information can be seen when the transaction is mined.
- The log is part of the blockchain and remain with the block.
- It is possible to subscribe to an event.
- Events are used with emit to manage writing to the log.

Structure of a Smart Contract

Ethereum Development: For Loops

- For loops have a construction like Java, C++, JavaScript etc.
- The for loop header consists of the initialization, the condition and the iterator
- The expression in the loop will execute as long as the header condition is true.

```
1  pragma solidity ^0.5.2;
2
3
4  contract newLoopContract {
5      function arith(uint a) public pure returns (uint b) {
6          b = 1;
7          for (uint i = 0; i < a; i++)
8              b = 2 * b + 100;
9      }
10 }
```

Structure of a Smart Contract

Ethereum Development: while loop

- The while loop has a header with a condition
while (k < toolsArray.length)
- The expression in the while loop's braces executes as long as the condition in the header is true.
- Do while loops are similar to while loops except the condition is evaluated at the end of the loop.
- Therefore, the loop expression will be executed at least once, even if the condition is false.

Structure of a Smart Contract

Ethereum Development: Handling Errors

- Given that Solidity is transaction based, it uses a state revert model when errors occur.
- This prevents wasteful transactions from occurring.
- Solidity does not currently support error catching.
- Structures such as “try catch” may be added in subsequent versions.

```
36 ▾ function issue (address receiver, uint amount) public {  
37  
38     uint256 j;  
39     require(receivedAssetsCnt > 0, "Received Asset Count must be great than 0");  
40 ▾     while (j < receivedAssetsCnt){  
41         require(msg.sender == issuer);  
42         require(amount < 1000);  
43         balance[receiver] += amount;  
44  
45         receivedAssetsCnt++;  
46     }  
47  
48 }  
49
```

Structure of a Smart Contract

Ethereum Development: revert

- The revert function also triggers exceptions to display an error.
- Revert will flag an error and reverse the call.
- An optional message can be passed into Revert to return error details.



```
[vm] from:0xca3...a733c to:MyContract.issue(address,uint256) 0xcbb...04d39 value:0 wei data:0x867...00064 logs:0 hash:0xdf1...64c5a
```

```
transact to MyContract.issue errored: VM error: revert.  
revert The transaction has been reverted to the initial state.  
Reason provided by the contract: "Received Asset Count must be great than 0".  Debug the transaction to get more information.
```

Structure of a Smart Contract

Summary

- The compiler version and pragma
- Comments and datatypes: integers, Booleans, strings addresses
- Enums and Structs
- Arrays and mappings
- Event and emits
- Loops
- Handling errors

Assessment

1. What is the command to declare the compiler version?
2. True or False: Contracts cannot have comments?
3. True or False: Are addresses a data type in contracts?
4. True or False: Enums are custom data types
5. True or False: Do structs model real-life entities?
6. True or False: Are arrays multivalued data types?
7. True or False: Are arrays only static?
8. True or False: Events capture certain data values?
9. True or False: emit does not write to the transaction log.
10. True or False: Loops run while a condition is false
11. True or False: Contracts cannot handle errors

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

Smart Contract Functions



coding-bootcamps.com

Thank you



Coding
Bootcamps