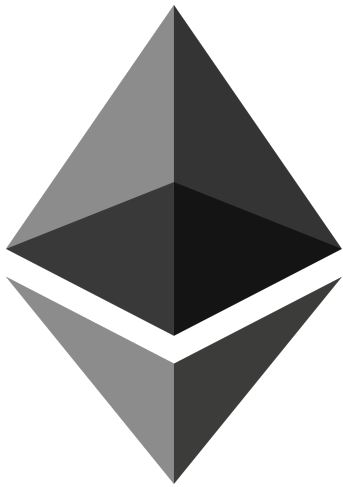




Ethereum and Smart contracts

Module 2 (part 2)

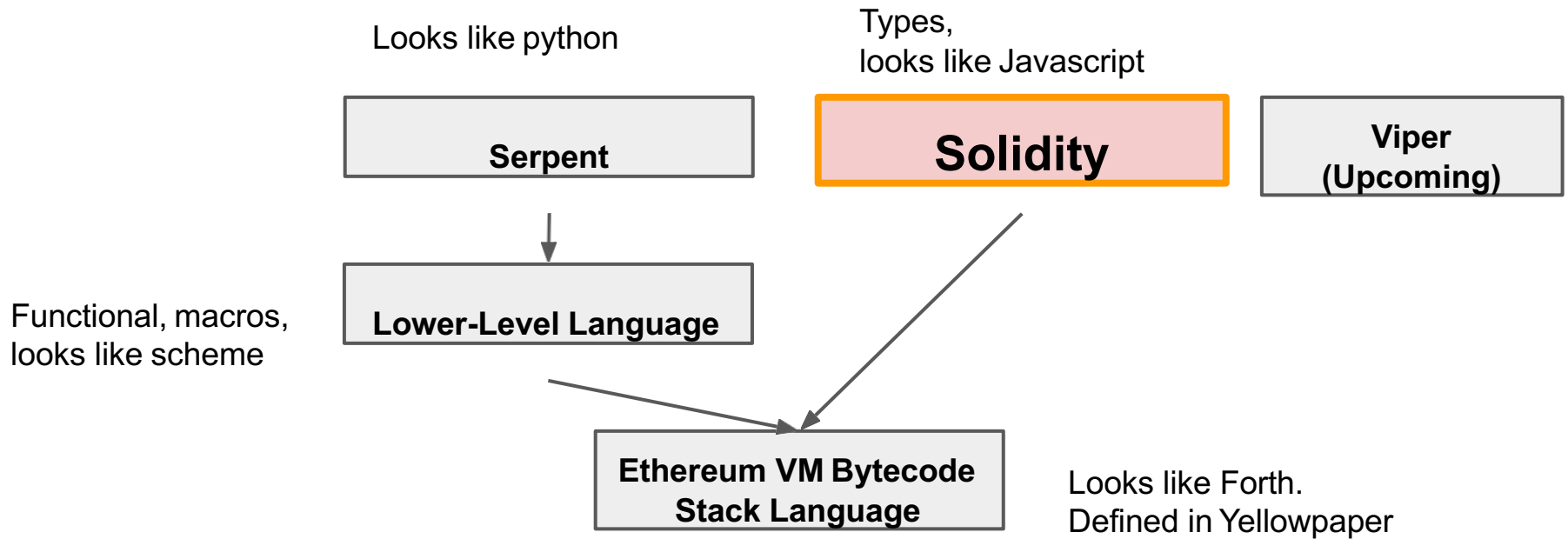
Part 2

Smart Contract Languages

- The most popular languages for writing smart contracts on Ethereum are:
- Solidity (the top pick)
- Vyper

Learning Solidity

Ethereum Languages



Writing smart contracts

Multiple high-level languages

- Solidity [<http://solidity.readthedocs.io>]
- Vyper [<https://github.com/ethereum/vyper>]

Single low-level language

- EVM code [<http://yellowpaper.io>]
- Stack-based

Solidity contract

```
Contract MyToken {  
    /* ... */  
}
```

Vyper contract

```
def register(key, value) {  
    /* ... */  
}
```



EVM code

```
PUSH 0x60  
PUSH 0x40  
MSTORE CALLDATASIZE  
ISZERO  
...  
PUSH2 0x1b4c  
POP  
JUMP
```

Which IDE to work with?

- To work with solidity and smart contracts, the best and easiest method I would suggest would be [Remix Solidity IDE](#) with [Metamask](#)
- If you just want an IDE in the browser for quick experimentation, you can also use [ethfiddle.com](#)
- *For local development you can use any editor,* there are plugins for [Atom](#), Pycharm, [Visual Studio Code](#), [Jetbrains IDEs](#)

Solidity should look familiar

- Syntax looks like JavaScript
- Contracts look like classes/objects

Main/basic components in Solidity

- ▶ Structure of a Smart Contract
- ▶ Types of variables
- ▶ Functions and modifiers
- ▶ Require, transfer, msg.sender and now
- ▶ Deploy a Smart Contract
- ▶ Use a Smart Contract in your web application

- Contracts in Solidity are similar to classes in **object-oriented languages**.
- Each contract can contain declarations of *State Variables, Functions, Function Modifiers, Events, Struct Types* and *Enum Types*.
- Furthermore, contracts can inherit from other contracts.

```
pragma solidity >=0.4.0 <0.6.0;
```

```
contract SimpleStorage {  
    uint storedData; // State variable  
    // ...  
}
```

Types of variables

- **State** variables are variables whose values are permanently stored in contract storage.
 - `bool`
 - `int` / `uint`: Signed and unsigned integers of various sizes. Keywords `uint8` to `uint256`
 - `address`: Holds a 20 byte value (size of an Ethereum address)
 - `address payable`: Same as `address`, but with the additional members `transfer` and `send`.
 - `string`
 - `bytes`
 - `mapping(_KeyType => _ValueType)`

`type [visibility] nameofvariable;`

Example of mapping

```
mapping (uint => string) phoneToName;
```

```
struct customer { unit idNum;  
                  string name;  
                  uint bidAmount;}
```

```
mapping (address => customer) custData;
```

Members of address

- balance, transfer, send(deprecated), call, delegatecall, staticcall

```
address payable x = address(0x123);  
address myAddress = address(this);  
if (x.balance < 10 && myAddress.balance >= 10) x.transfer(10);
```

addr.call.value(amount) is essentially the same as
addr.transfer(amount)

Function and modifiers

- Function types come in two flavors - **internal** and **external** functions
- Internal functions can only be called inside the current contract (by default)--do not create an actual EVM call (also called a “message call”)
- External functions consist of an address and a function signature and they can be called from other contracts and via transactions -- do create an actual EVM call

General form of a function

```
function name(<parameter types>) {public|private|..  
{internal|external} {pure|view|payable} returns (<return types>){  
  
}
```

view functions are functions, that are promised not to modify the state.

pure functions are functions, that are promised not to modify or read the state.

payable allows a function to receive ether while being called

```
pragma solidity ^0.4.16;  
  
contract cont {  
    function func(uint x, uint y) view returns (uint) {  
        return x * (y + 42) + now;  
    }  
}
```

Visibility modifiers

- `public, private, internal, external`
- Applied to both functions and variables

(For state variables, external is not possible and the default is internal)

// This will not compile

```
pragma solidity ^0.4.0;
```

```
contract C {  
    uint private data;  
  
    function f(uint a) private returns(uint b) { return a + 1; }  
    function setData(uint a) public { data = a; }  
    function getData() public returns(uint) { return data; }  
    function compute(uint a, uint b) internal returns (uint) { return a+b; }  
}
```

```
contract D {  
    function readData() public {  
        C c = new C();  
        uint local = c.f(7); // error: member `f` is not visible  
        c.setData(3);  
        local = c.getData();  
        local = c.compute(3, 5); // error: member `compute` is not visible  
    }  
}
```

```
contract E is C {  
    function g() public {  
        C c = new C();  
        uint val = compute(3, 5); // access to internal member (from derived to parent contract)  
    }  
}
```

Fall back function

- A contract can have exactly one unnamed function. This function cannot have arguments, cannot return anything and has to have external visibility.
- <https://solidity.readthedocs.io/en/develop/contracts.html#fallback-function>

Constructors

- When a contract is created, its constructor (`constructor` keyword) is executed once. A constructor is optional. Only one constructor is allowed, and this means overloading is not supported.

Activity: Compile and run a Hello World contract

- Use Ethereum Remix
(<https://remix.ethereum.org>)

GITTER

Where communities thrive

FREE FOR COMMUNITIES

JOIN OVER 800K+ PEOPLE

JOIN OVER 90K+ COMMUNITIES

CREATE YOUR OWN COMMUNITY

EXPLORE MORE COMMUNITIES

Browser, Desktop and Mobile Apps.



ethereum/solidity The Solidity Contract-Oriented Programming Language - By chatting here you agree...



contract (preferably on chain)?



Mike Shultz @mikeshultz

Jan 27 18:00

Why would `return uint(-1)` return max value uint256 instead of `-1` ? Something to do with `-1` actually being `int_const` ? There any way to return a literal uint256 -1?



ccolorado @ccolorado

Jan 27 18:04

@mikeshultz what do you have on you function's returns(TYPE)?



Mike Shultz @mikeshultz

Jan 27 18:04

`returns (uint)`



ccolorado @ccolorado

Jan 27 18:06

your return is casting -1 as a uint, that If I am not mistaken thats how strongly typed messages handle it

you could change your return type to int256 and get a -1

uint can only return non signed iteger, that would exclude the non int constant you are suggesting

did I answer **your** question, or just **my** version of your question ? :)



Mike Shultz @mikeshultz

Jan 27 18:09

Yeah, I think you just shook loose some cobwebs. Of course uint is *unsigned*... Maybe I need a break...

Thank you



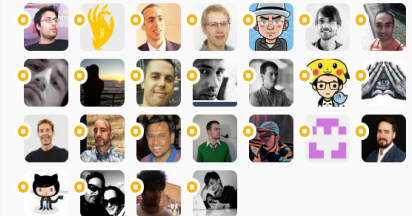
ccolorado @ccolorado

Jan 27 18:10

you and I

SIGN IN TO START TALKING

PEOPLE REPO INFO



SEE ALL (6293 PEOPLE)

ACTIVITY

@jpitts banned

Oct 10 2017

@etherchamp1_twitter

@chriseth banned

Jun 05 2016

@adamskee



<https://gitter.im/ethereum/solidity>

A “dumb contract” example

Alice will reveal to Bob a value x such that
 $\text{SHA-256}(x) = 0x2a\dots$

In exchange, Bob will give Alice \$10 in cash.

If Alice does not give Bob by July 1, 2018,
then she will pay a penalty of US\$1 per day
that she is late, up to US\$100.

Signed: *Alice Bob*

Exercise: Complete the code

```
contract PayForHash {  
  
    address alice = 0xaaa; // address of alice  
    address bob = 0xbbb; // address of bob  
    uint deadline = 0; // set deadline to July 1st  
    bool hashSent = false;  
  
    function sendData(bytes x) public {  
        require (msg.sender == alice);  
        if (sha3(x) == '0xdeadbeef') {  
            hashSent = true;  
            alice.transfer(10);  
        }  
    }  
    function bobClaim() public {  
        // send money to Bob if Alice has not  
        // submitted the hash and deadline has  
        // passed  
    }  
    function getDelay() public {  
        // return # of days past the deadline  
    }  
}
```

Special/global variables and functions

- There are special variables and functions that provide **block** or **transactions** related information to help developers implement their logics.
- `msg` (together with `tx` and `block`) is a special global variable that contains some properties which allow access to the blockchain.

Special variables and functions

- `msg.sender`: return the address of sender of the message/call -- is always the address where the current (external) function call came from [the person/contract who is currently connecting with the contract]
- `msg.value`: returns number of wei sent with the message/call in the transaction
[When you send ETH to contract, `msg.value` will be equal to the amount sent]
- `now`: current block timestamp (alias for `block.timestamp`)
- `require(bool condition)`: abort execution and revert state changes if condition is false (use for malformed input or error in external component)
- `sha256(bytes memory)` returns `(bytes32)`: compute the SHA-256 hash of the input
- `tx.gasprice`: gas price of the transaction
- ...

```
function f(uint start, uint daysAfter) public {  
    if (now >= start + daysAfter * 1 days) {  
        // ...  
    }  
}
```

Check out solidity [doc](#)

Example smart contract

Contract name

```
contract SimpleBank {
```

```
    mapping(address => uint) balances;
```

```
    function deposit(uint amount) {  
        balances[msg.sender] += amount;  
    }
```

```
    function withdraw() {  
        msg.sender.call.value(balances[msg.sender])();  
        balances[msg.sender] = 0;  
    }  
}
```

Example smart contract

Local state maintains a mapping from user addresses to unsigned integers

```
contract SimpleBank {  
    mapping(address => uint) balances;  
  
    function deposit(uint amount) {  
        balances[msg.sender] += amount;  
    }  
  
    function withdraw() {  
        msg.sender.call.value(balances[msg.sender])();  
        balances[msg.sender] = 0;  
    }  
}
```

Example smart contract

```
contract SimpleBank {  
    mapping(address => uint)
```

Function that allows users to deposit ether at the SimpleBank contract

```
    function deposit(uint amount) {  
        balances[msg.sender] += amount;  
    }
```

msg.sender returns the address of the transaction sender

```
        balances[msg.sender]();  
    }
```

```
}
```

Example smart contract

```
contract SimpleBank {
```

```
    mapping(address => uint256)
```

```
    function deposit(uint256 amount) public {
        balances[msg.sender] += amount;
    }
}
```

Trigger a transfer of ETH with value equal to **balances[msg.sender]** to the transaction sender

```
    function withdraw() public {
        msg.sender.call.value(balances[msg.sender])();
        balances[msg.sender] = 0;
    }
}
```

Set the balance of the transaction sender to 0

Another example

```
pragma solidity ^0.4.21;

contract Coin {
    // The keyword "public" makes those variables
    // readable from outside.
    address public minter;
    mapping (address => uint) public balances;

    // Events allow light clients to react on
    // changes efficiently.
    event Sent(address from, address to, uint amount);

    // This is the constructor whose code is
    // run only when the contract is created.
    function Coin() public {
        minter = msg.sender;
    }

    function mint(address receiver, uint amount) public {
        if (msg.sender != minter) return;
        balances[receiver] += amount;
    }

    function send(address receiver, uint amount) public {
        if (balances[msg.sender] < amount) return;
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}
```


Clever implementation of maps in Solidity

```
mapping(string => uint256) balances;
```

Alice	15
Bob	15
Joe	100



- every item requires at least one 256-bit word
- `balances["Andrew"]` is 0 if "Andrew" doesn't exist or if "Andrew" has 0 balance
- to delete a key, set `balances["Andrew"] = 0`
- Cannot delete an entire map!

Syntax you need to know, references

- **Storage and stateful methods**

 - public instance variables, public methods*

 - constant, pure, view methods*

 - <https://solidity.readthedocs.io/en/v0.4.21/contracts.html#functions>

 - <https://solidity.readthedocs.io/en/v0.4.21/contracts.html#visibility-and-getters>

- **Control flow, for loops and if statements**

 - <https://solidity.readthedocs.io/en/v0.4.21/control-structures.html#control-structures>

Syntax you need to know, references

- Events and printf debugging

event Debug(string); ... emit Debug("fail at point A")

<https://solidity.readthedocs.io/en/v0.4.21/contracts.html#events>

Debugging strategies:

1. Use Log Events
2. Use pure functions
3. Use the Remix debugger

Syntax you need to know, references

- **Declaring variables and conversion between types**

<https://solidity.readthedocs.io/en/v0.4.21/types.html#value-types>

<https://solidity.readthedocs.io/en/v0.4.21/types.html#conversions-between-elementary-types>

- **Integers can overflow**

<https://github.com/OpenZeppelin/zeppelin-solidity/blob/master/contracts/math/SafeMath.sol>

- **Mappings**

mapping (address => bool) hasAlreadyVoted;

<https://solidity.readthedocs.io/en/v0.4.21/types.html#mappings>

Syntax you need to know, references

- **Payable and transferring currency**

address x; x.transfer(msg.value)

<https://solidity.readthedocs.io/en/v0.4.21/types.html#address>

<https://solidity.readthedocs.io/en/v0.4.21/units-and-global-variables.html#address-related>

- **Arrays in storage and in memory**

<https://solidity.readthedocs.io/en/v0.4.21/types.html#reference-types>

<https://solidity.readthedocs.io/en/v0.4.21/types.html#arrays>

Syntax you need to know, references

- **Calling methods of other contracts, the Gas model**

Tx.gas, gasLeft()

<https://solidity.readthedocs.io/en/v0.4.21/units-and-global-variables.html#special-variables-and-functions>

- **Extern / abstract contracts**

<https://solidity.readthedocs.io/en/v0.4.21/contracts.html#abstract-contracts>

- **Access controls**

msg.sender, tx.origin

<https://solidity.readthedocs.io/en/v0.4.21/common-patterns.html?highlight=payable#restricting-access>

Solidity gotchas (many more later)

- Member variables can be marked public
 - Getter methods automatically provided
- **Functions must be marked payable to accept funds**
- Member variables go to storage by default
 - Method variables go to memory
- Fallback function()
 - Called if no function specified (e.g. send)
 - Called if non-existent function called
- **msg.sender** vs. **tx.origin**

<https://solidity.readthedocs.io/en/develop/solidity-in-depth.html>

- **Semantic versioning** prevents upgrades that would cause an API contract to break.
- Example: **pragma solidity** >=0.4.0 <0.6.0;
 - tells that the source code is written for Solidity version 0.4.0 or anything newer that does not break functionality (up to, but not including, version 0.6.0). This is to ensure that the contract is not compilable with a new (breaking) compiler version, where it could behave differently
 - **pragma solidity** ^0.4.0;
 - source file will not compile with a compiler earlier than version 0.4.0 and it will also not work on a compiler starting from version 0.5.0 (this second condition is added by using ^).

Activity: run blockchain-based puzzle game

- [Link](#) to the code
- In this game there are six levels, each with their own puzzle to solve. The first user to correctly solve all of the puzzles in order is awarded a sum of ETH.
- (Do it at home)