FTEC 4004

E-payment Systems and Cryptocurrency Technologies

Tutorial 11

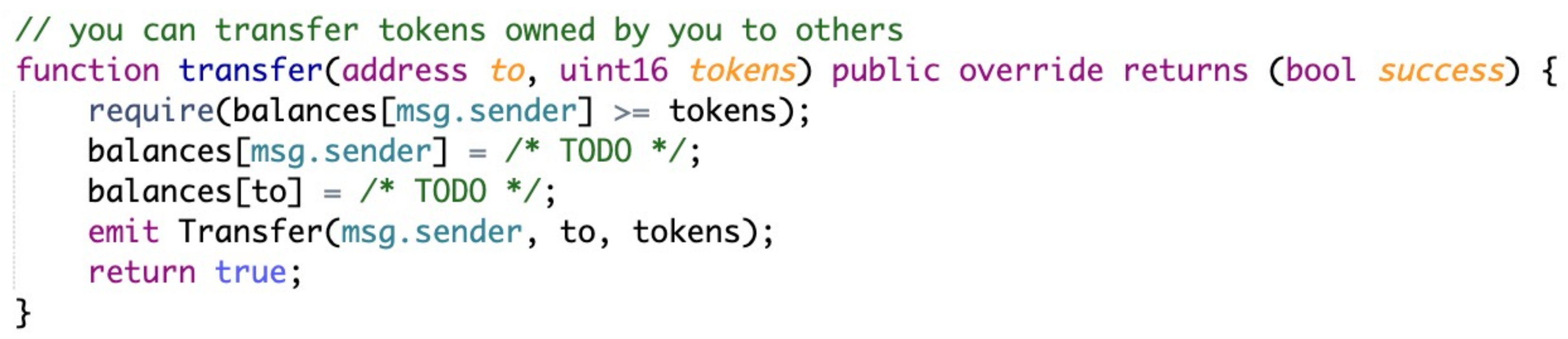
To Start a Journey as a Smart Contract Programmer

WANG Xianbo

xianbo@ie.cuhk.edu.hk

Homework 5 is released

- Build your own token and start your crowdsale!
- Only need to be able to read and understand Solidity code.
- Don’t need to write the whole code: “*fill in the blanks*”



```
// you can transfer tokens owned by you to others
function transfer(address to, uint16 tokens) public override returns (bool success) {
    require(balances[msg.sender] >= tokens);
    balances[msg.sender] = /* TODO */;
    balances[to] = /* TODO */;
    emit Transfer(msg.sender, to, tokens);
    return true;
}
```

Overview of today’s topic

- Resources that can help you become a smart contract programmer
- A basic intro of Solidity programming language
- Demo: Helloworld Smart Contract

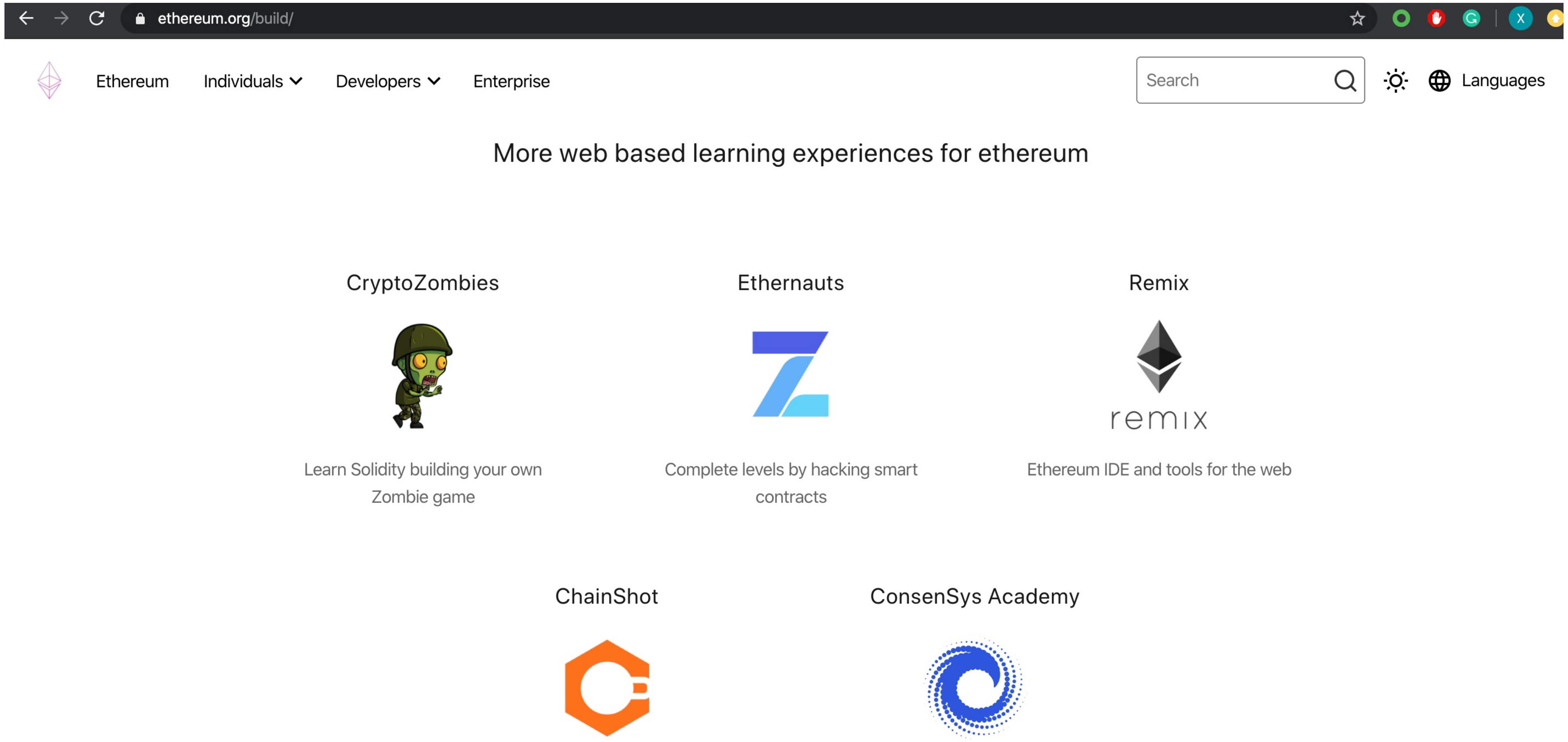
Smart Contract Developer?

Personal opinion:

- Always good to master one more programming language.
- Not so many people know smart contract coding. Can differenciate yourself in the job market.
- It’s also a “risky” path. Most smart contract companies/projects are at small scale or are start-ups. It may not last long.
- Blockchain techniques evolve fast, you need to learn fast.
- The “golden age” has gone. Maybe there will be a 2nd wave?

* Smart Contract Developer Salaries and Future Growth?: https://www.reddit.com/r/ethdev/comments/9mihiu/smart_contract_developer_salaries_and_future/

Cut the … Where can I start?



ethereum.org/build/

Ethereum Individuals Developers Enterprise Search Languages

More web based learning experiences for ethereum

CryptoZombies

Learn Solidity building your own Zombie game

Ethernauts

Complete levels by hacking smart contracts

Remix

Ethereum IDE and tools for the web

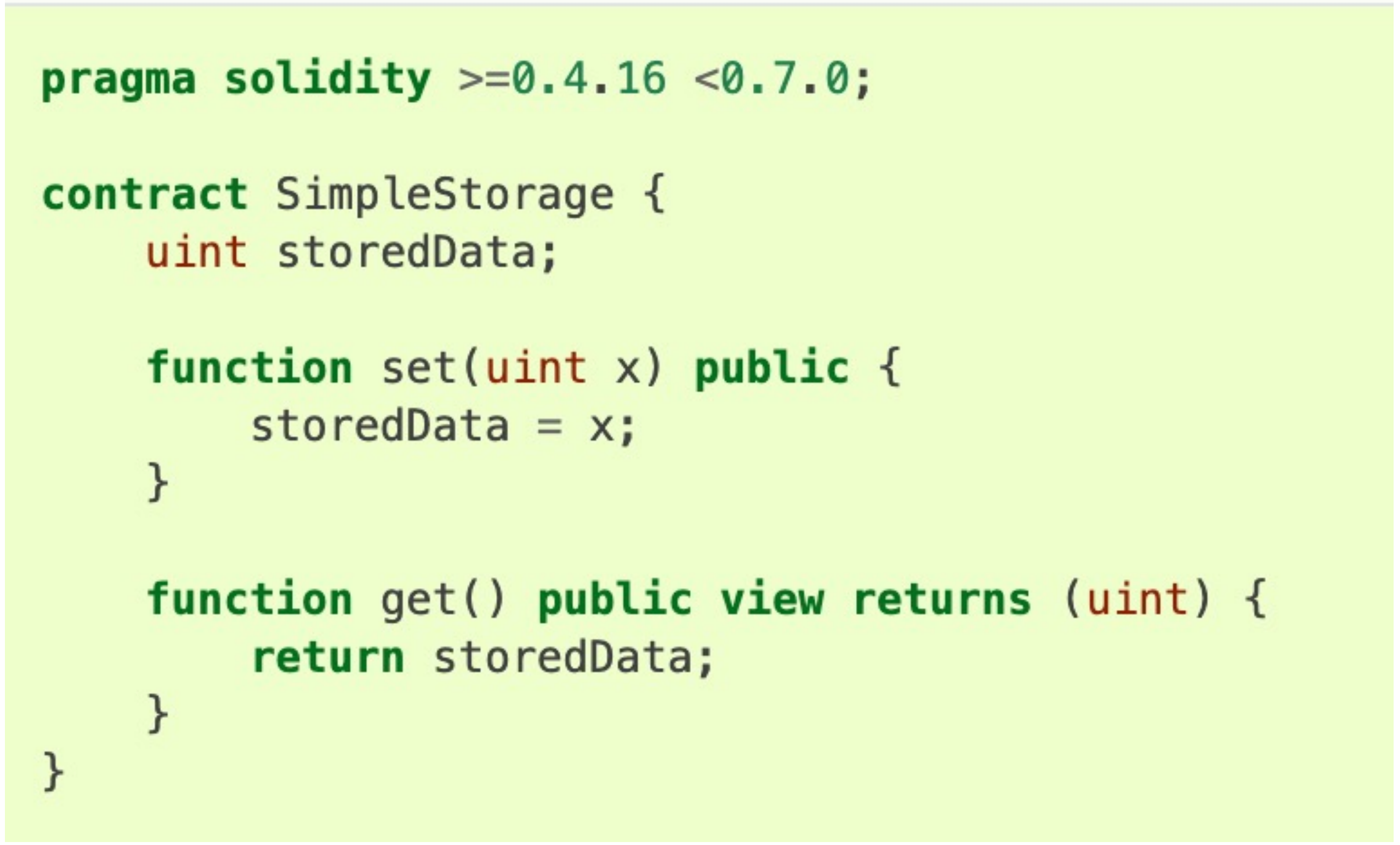
ChainShot

ConsenSys Academy

Quick Intro for Solidity Programming

What does it look like?

- JavaScript-alike
- Object-oriented
- Statically-typed



```
pragma solidity >=0.4.16 <0.7.0;

contract SimpleStorage {
    uint storedData;

    function set(uint x) public {
        storedData = x;
    }

    function get() public view returns (uint) {
        return storedData;
    }
}
```

Version Pragma

- First line of a smart contract, declare expected compiler versions

```
pragma solidity >=0.4.16 <0.7.0;
```

- The compiler will issue an error if the version checking fails.
- Solidity standards are moving fast, major versions (e.g. 0.6.x and 0.5.x) have many differences that are not backwards-compatible.
- This tutorial is based on 0.6.6 (latest is 0.8.4, it evolves fast)

> When deploying contracts, you should use the latest released version of Solidity. This is because breaking changes as well as new features and bug fixes are introduced regularly. We currently use a 0.x version number [to indicate this fast pace of change](https://semver.org/#spec-item-4).

Import & Comments

- As many other languages, you can import other source code into current source. Useful for modularizing your code

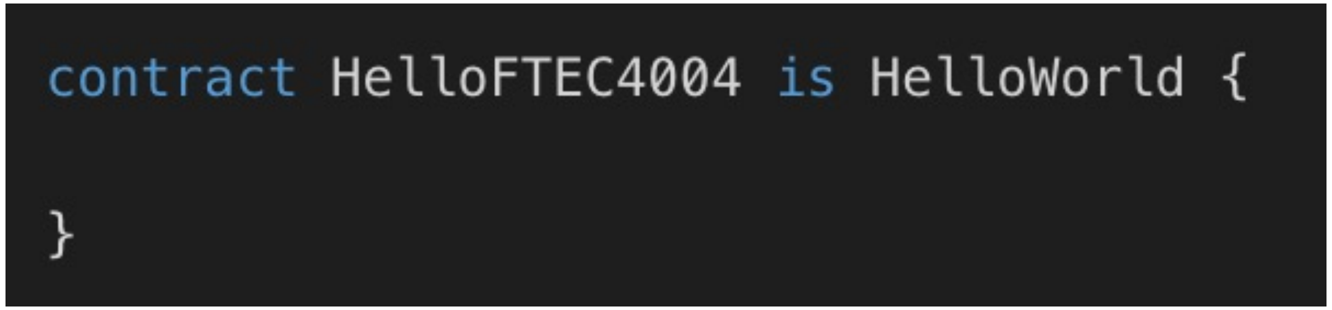
```
import "filename";
```

- Single line and multiline comments (same as C/C++)

```
// This is a single-line comment.

/*
This is a
multi-line comment.
*/
```

Contract

- Like class in most object-oriented languages.
- Each contract can contain declarations of **State Variables**, **Functions**, Function Modifiers, **Events**, Struct Types and Enum Types.
- Contracts can inherit from other contracts.

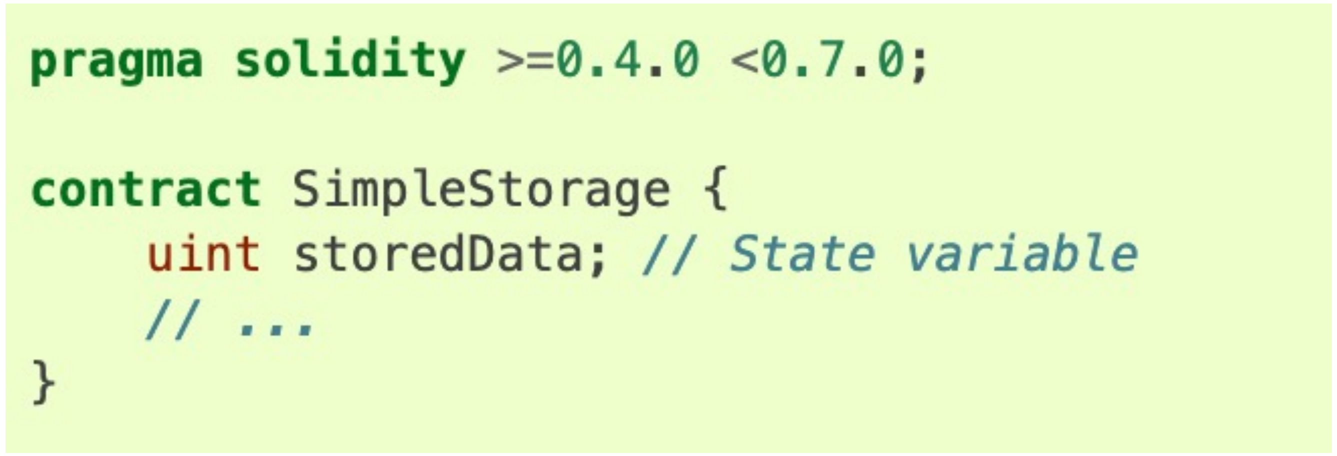


```
contract HelloFTEC4004 is HelloWorld {

}
```

State Variables

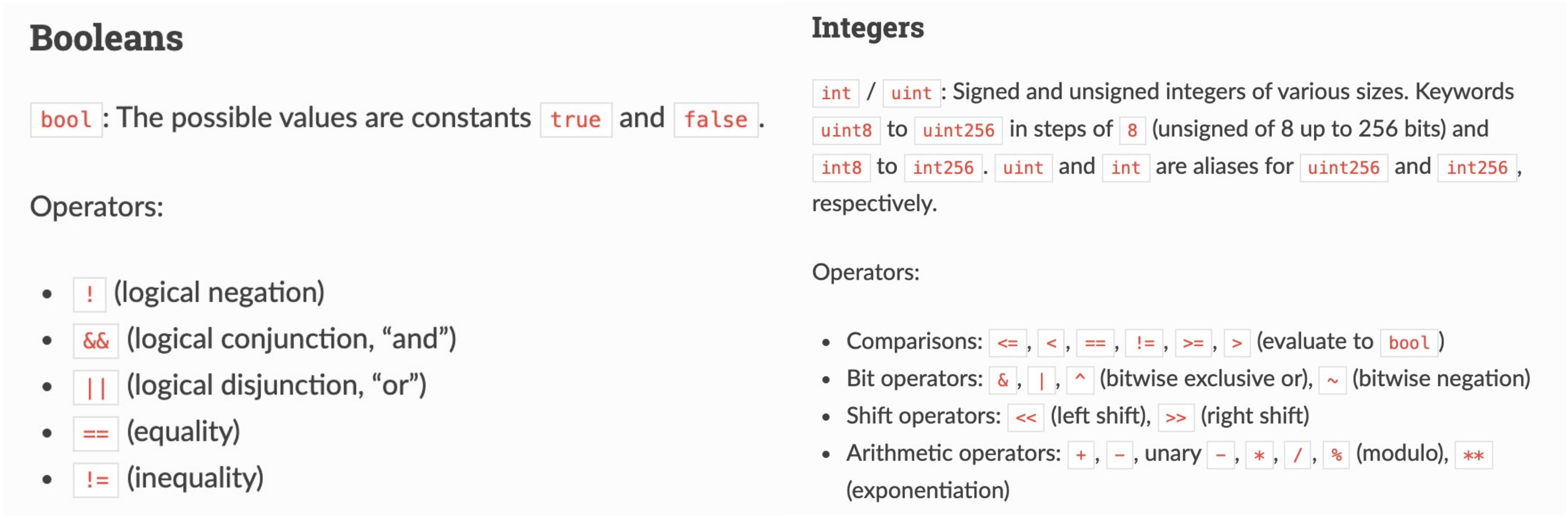
- State variables are permanently stored in contract storage

- Smart Contract maintains a key-value storage.
- Ethereum blockchain store the most recent state of each contract
- Transactions between contracts cause state change



```
pragma solidity >=0.4.0 <0.7.0;

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

Data Types



Booleans

`bool`: The possible values are constants `true` and `false`.

Operators:

- `!` (logical negation)
- `&&` (logical conjunction, “and”)
- `||` (logical disjunction, “or”)
- `==` (equality)
- `!=` (inequality)

Integers

`int` / `uint`: Signed and unsigned integers of various sizes. Keywords `uint8` to `uint256` in steps of `8` (unsigned of 8 up to 256 bits) and `int8` to `int256`. `uint` and `int` are aliases for `uint256` and `int256`, respectively.

Operators:

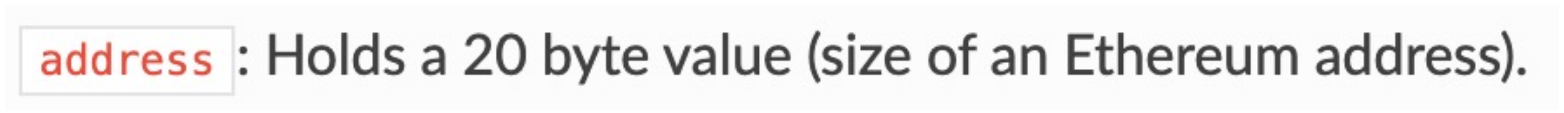
- Comparisons: `<=`, `<`, `==`, `!=`, `>=`, `>` (evaluate to `bool`)
- Bit operators: `&`, `|`, `^` (bitwise exclusive or), `~` (bitwise negation)
- Shift operators: `<<` (left shift), `>>` (right shift)
- Arithmetic operators: `+`, `-`, unary `-`, `*`, `/`, `%` (modulo), `**` (exponentiation)

Overflow/Underflow: `uint256(0) - uint256(1) == 2**256 - 1`

Division returns integer: `int256(-5) / int256(2) == int256(-2)`

Data Types

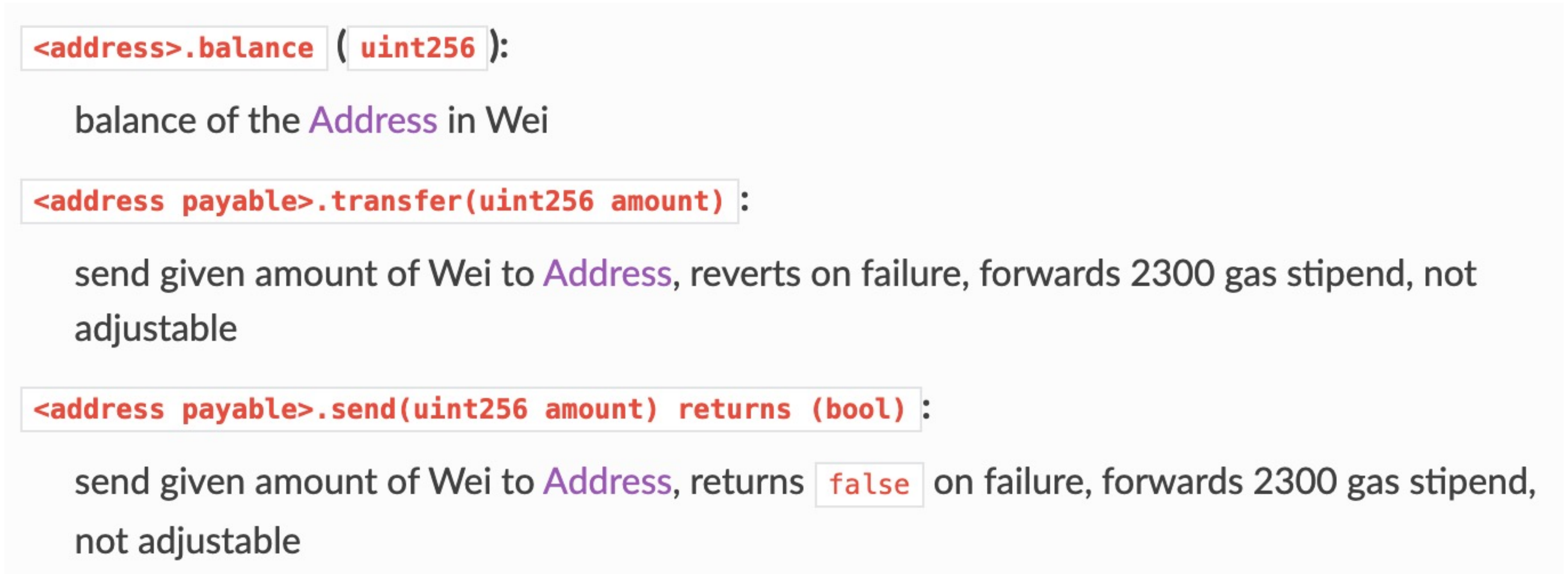
Address



`address`: Holds a 20 byte value (size of an Ethereum address).

Value e.g. 0xE0f5206BBD039e7b0592d8918820024e2a7437b9

Operations



`<address>.balance` (`uint256`):

balance of the Address in Wei

`<address payable>.transfer(uint256 amount)`:

send given amount of Wei to Address, reverts on failure, forwards 2300 gas stipend, not adjustable

`<address payable>.send(uint256 amount) returns (bool)`:

send given amount of Wei to Address, returns `false` on failure, forwards 2300 gas stipend, not adjustable

More Data Types

Structs

```
struct Voter { // Struct
    uint weight;
    bool voted;
    address delegate;
    uint vote;
}
```

Arrays

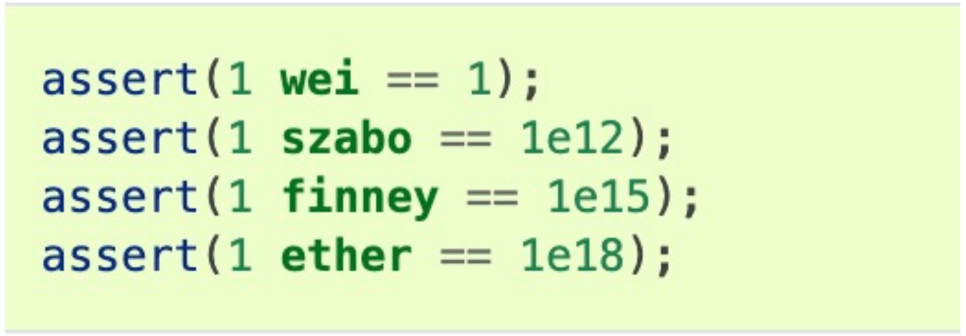
```
uint[] x;
Voter[] voters;
x.push(3);
int count = voters.length;
```

Mappings (hash tables)

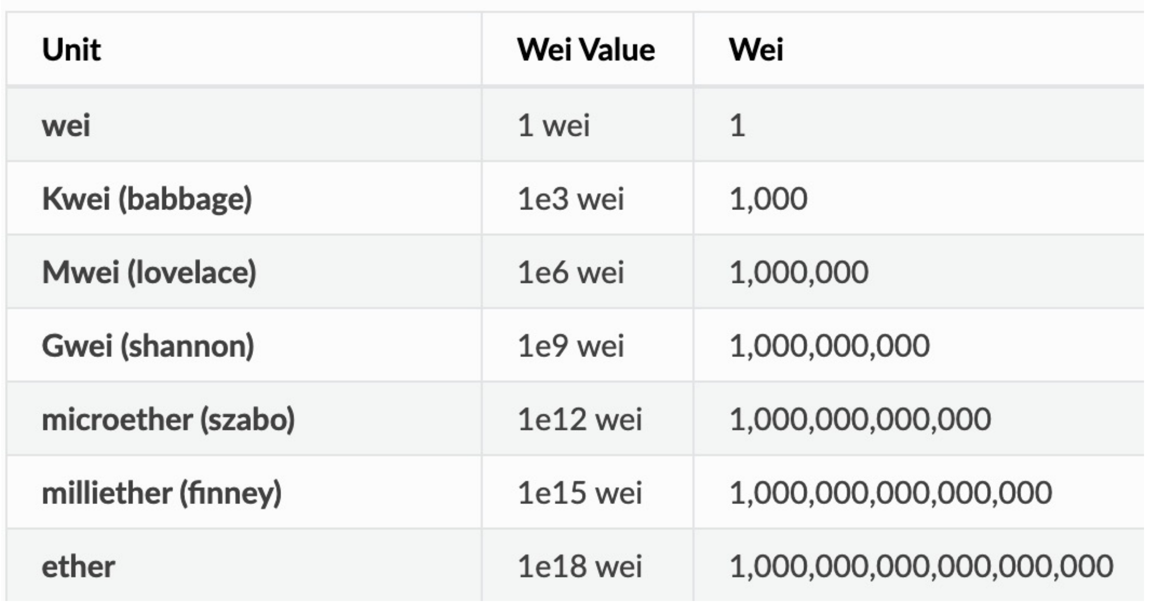
```
mapping (address => uint256) private _balances;
mapping (address => mapping (address => uint256)) private _allowances;
```

Units

Supported units in Solidity

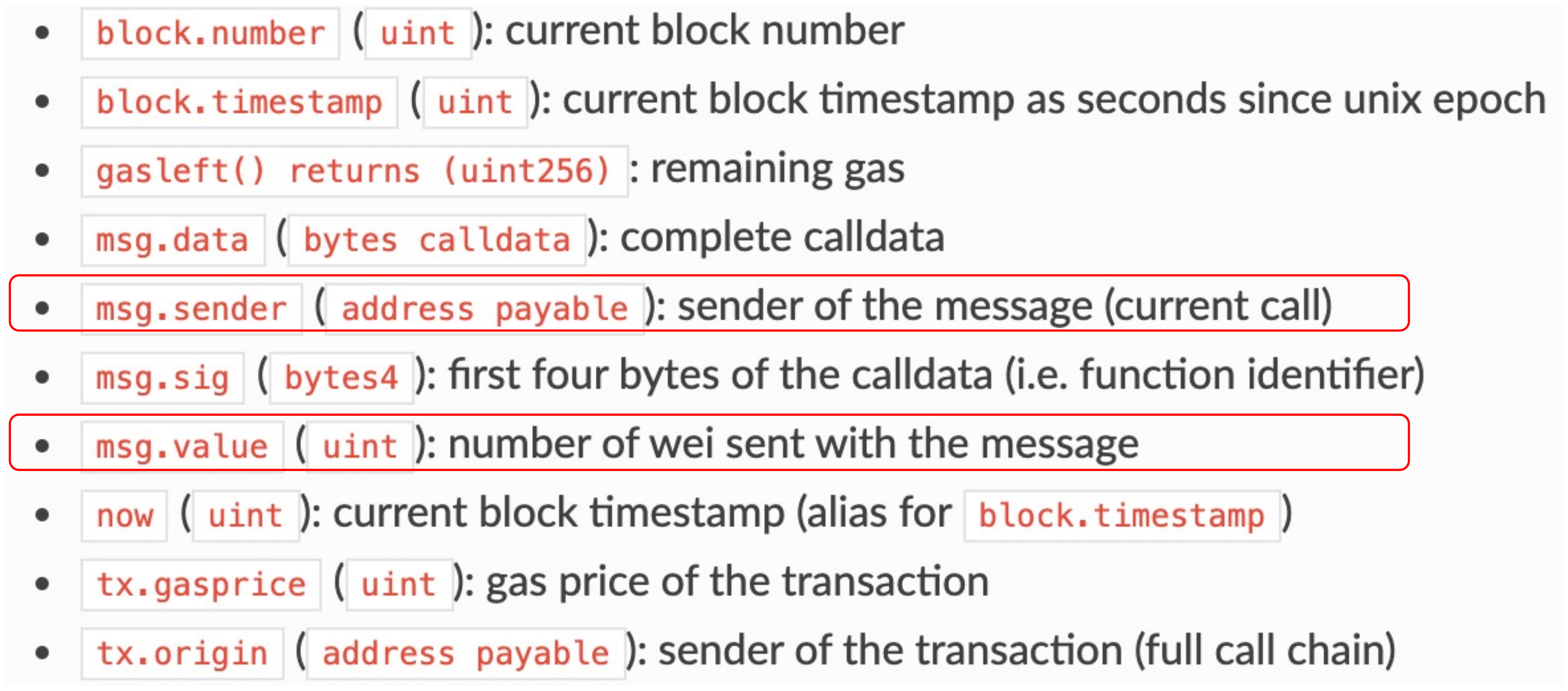


```
assert(1 wei == 1);
assert(1 szabo == 1e12);
assert(1 finney == 1e15);
assert(1 ether == 1e18);
```



Unit	Wei Value	Wei
wei	1 wei	1
Kwei (babbage)	1e3 wei	1,000
Mwei (lovelace)	1e6 wei	1,000,000
Gwei (shannon)	1e9 wei	1,000,000,000
microether (szabo)	1e12 wei	1,000,000,000,000
milliether (finney)	1e15 wei	1,000,000,000,000,000
ether	1e18 wei	1,000,000,000,000,000,000

Some Special Variables



- `block.number` (`uint`): current block number
- `block.timestamp` (`uint`): current block timestamp as seconds since unix epoch
- `gasleft() returns (uint256)`: remaining gas
- `msg.data` (`bytes calldata`): complete calldata
- `msg.sender` (`address payable`): sender of the message (current call)
- `msg.sig` (`bytes4`): first four bytes of the calldata (i.e. function identifier)
- `msg.value` (`uint`): number of wei sent with the message
- `now` (`uint`): current block timestamp (alias for `block.timestamp`)
- `tx.gasprice` (`uint`): gas price of the transaction
- `tx.origin` (`address payable`): sender of the transaction (full call chain)

Functions

- Functions can be invoked **externally** (by wallets or by other contracts) or **internally** (by code in the same contract)

- Functions definition includes arguments and returns.

```
pragma solidity >=0.4.0 <0.7.0;

contract SimpleAuction {
    function bid() public payable { // Function
        // ...
    }
}
```

```
pragma solidity >=0.4.16 <0.7.0;

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

Function Modifiers

- function (<parameter types>) {internal|external} [pure|constant|view|payable] [returns (<return types>)]

- function types are by default internal
- contract functions are by default public

- Visibility summary:
- *public* - all can access
- *external* - cannot be accessed internally, only externally
- *internal* - only this contract and contracts deriving from it can access
- *private* - can be accessed only from this contract
- (each classifier classifies a subset of the former)

Visibility of Functions

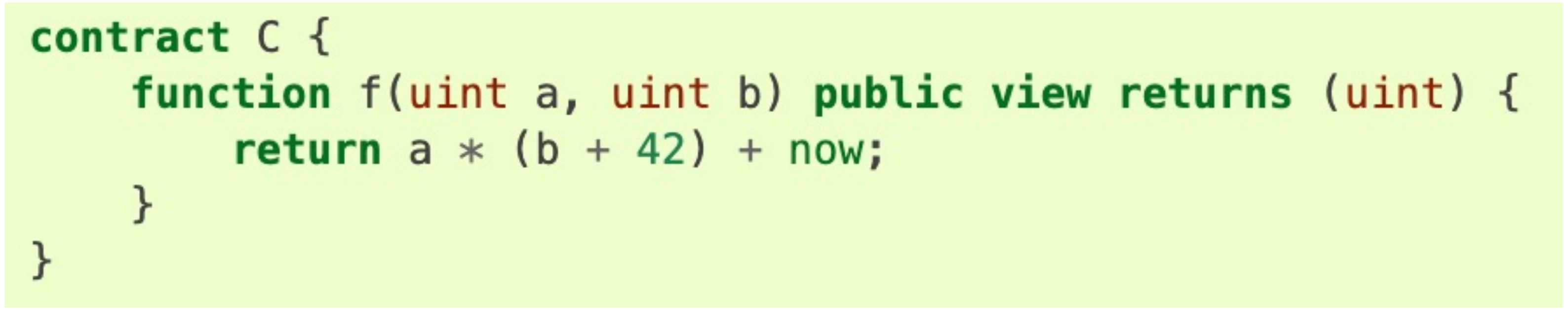
- *Public* functions can be either called internally or via messages.
- *External* functions can be called from other contracts and via tx’s.
  - they cannot be called internally (i.e. f() does not work, but *this*.f() works).

- *Internal* functions can only be accessed internally (from within the current contract or contracts deriving from it), without using *this*.

- *Private* functions are only visible for the contract they are defined in and not in derived contracts.

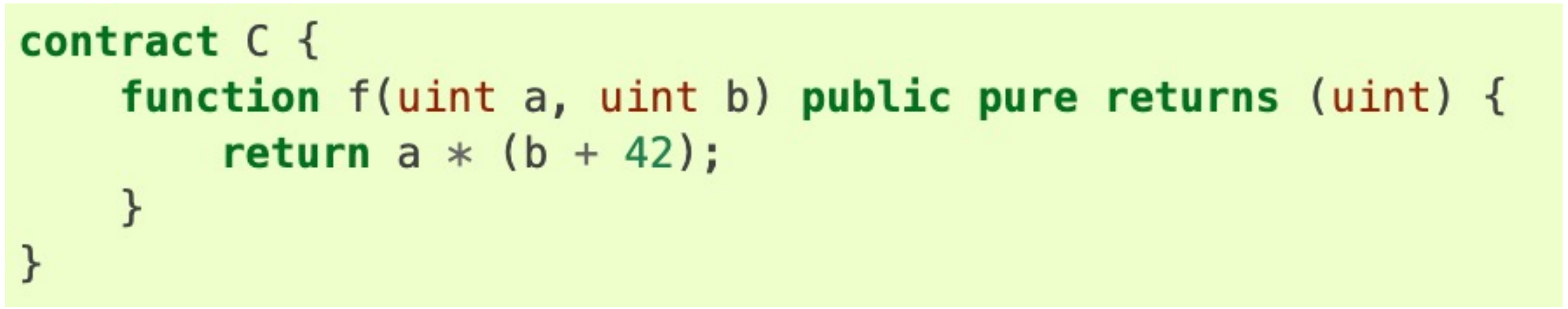
Function Types

- View function: not modifying state, but only read state variables



```
contract C {
    function f(uint a, uint b) public view returns (uint) {
        return a * (b + 42) + now;
    }
}
```

- Pure function: neither read nor modify state variables

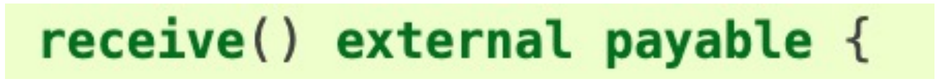


```
contract C {
    function f(uint a, uint b) public pure returns (uint) {
        return a * (b + 42);
    }
}
```

Function Types

- Payble Function: only function with payble modifier will accept ether transaction.

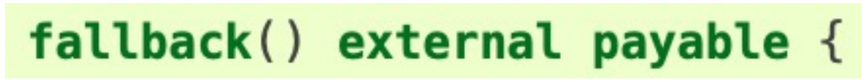
- Receive Ether Function



```
receive() external payable {
```

  - The function that is executed on plain Ether transfers (e.g. via *.send()* or *.transfer()*).
  - At most one per contract. No arguments, no returns.

- Fallback Function



```
fallback() external payable {
```

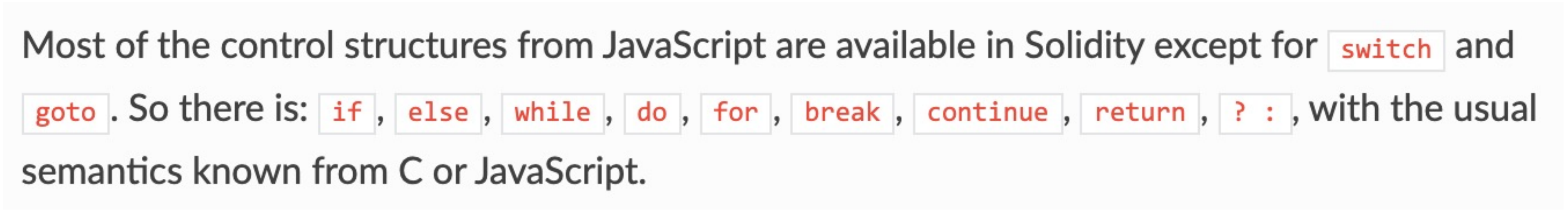
  - Executed on a call if no other matching functions.
  - At most one per contract. No arguments, no returns.

Events

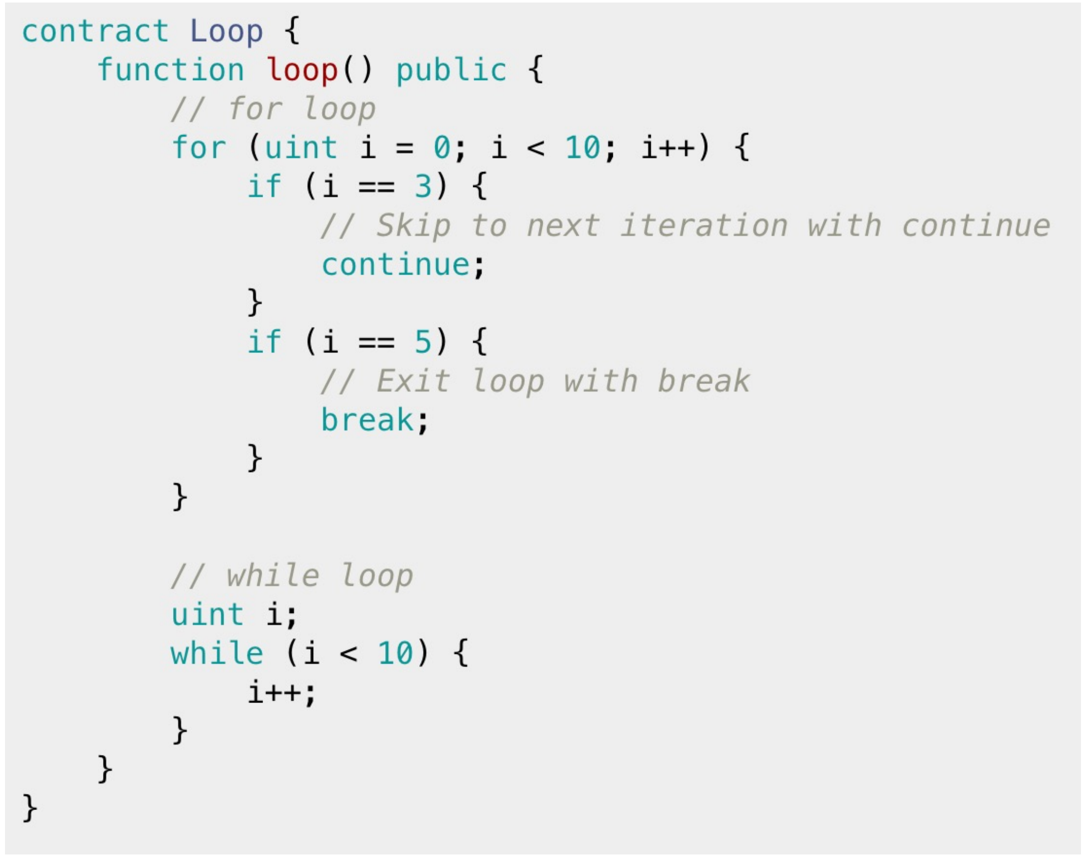
Logging in Solidity are implemented with events.

```
pragma solidity >=0.4.21 <0.7.0;

contract SimpleAuction {
    event HighestBidIncreased(address bidder, uint amount); // Event

    function bid() public payable {
        // ...
        emit HighestBidIncreased(msg.sender, msg.value); // Triggering event
    }
}
```

Control Structures



Most of the control structures from JavaScript are available in Solidity except for `switch` and `goto`. So there is: `if`, `else`, `while`, `do`, `for`, `break`, `continue`, `return`, `? :`, with the usual semantics known from C or JavaScript.



```
contract Loop {
    function loop() public {
        // for loop
        for (uint i = 0; i < 10; i++) {
            if (i == 3) {
                // Skip to next iteration with continue
                continue;
            }
            if (i == 5) {
                // Exit loop with break
                break;
            }
        }

        // while loop
        uint i;
        while (i < 10) {
            i++;
        }
    }
}
```

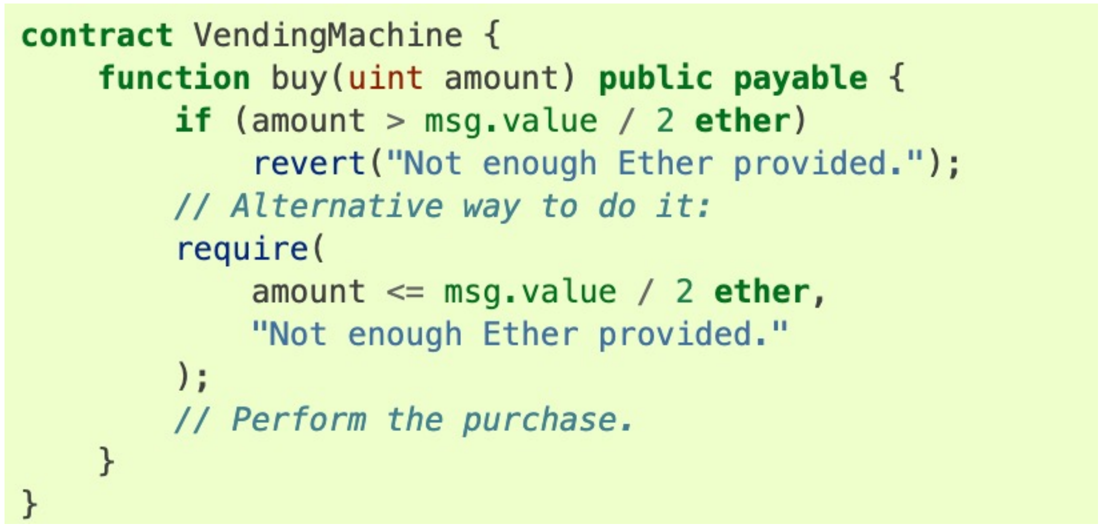
Error Handling

require

The `require` function should be used to ensure valid conditions that cannot be detected until execution time. This includes conditions on inputs or return values from calls to external contracts.

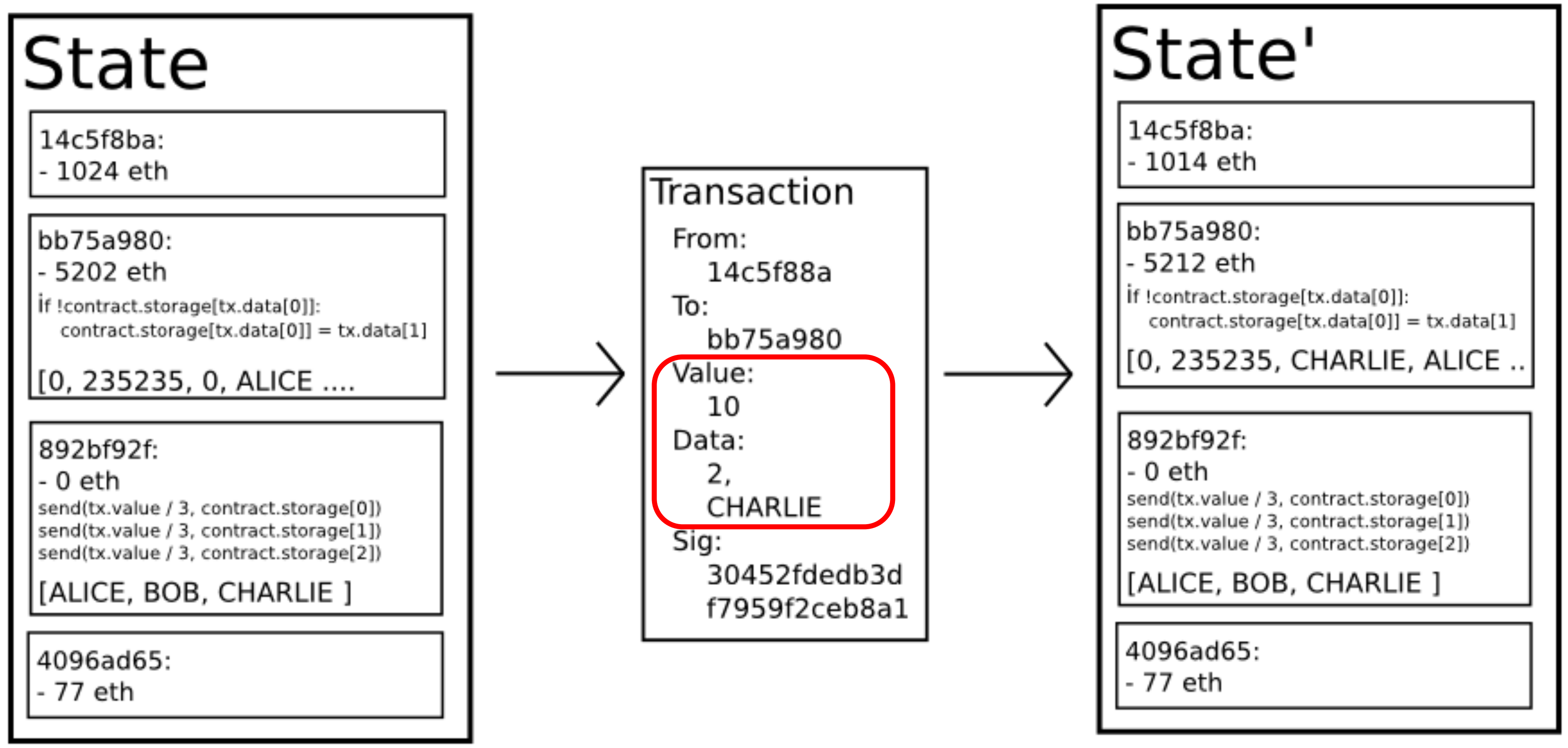
revert

The `revert` function is another way to trigger exceptions from within other code blocks to flag an error and revert the current call. The function takes an optional string message containing details about the error that is passed back to the caller.



```
contract VendingMachine {
    function buy(uint amount) public payable {
        if (amount > msg.value / 2 ether)
            revert("Not enough Ether provided.");
        // Alternative way to do it:
        require(
            amount <= msg.value / 2 ether,
            "Not enough Ether provided."
        );
        // Perform the purchase.
    }
}
```

Low Level Data Structure


State
14c5f8ba:
- 1024 eth
bb75a980:
- 5202 eth
if !contract.storage[tx.data[0]]:
contract.storage[tx.data[0]] = tx.data[1]
[0, 235235, 0, ALICE ....
892bf92f:
- 0 eth
send(tx.value / 3, contract.storage[0])
send(tx.value / 3, contract.storage[1])
send(tx.value / 3, contract.storage[2])
[ALICE, BOB, CHARLIE]
4096ad65:
- 77 eth
Transaction
From:
14c5f88a
To:
bb75a980
Value:
10
Data:
2,
CHARLIE
Sig:
30452fdedb3d
f7959f2ceb8a1
State'
14c5f8ba:
- 1014 eth
bb75a980:
- 5212 eth
if !contract.storage[tx.data[0]]:
contract.storage[tx.data[0]] = tx.data[1]
[0, 235235, CHARLIE, ALICE ..
892bf92f:
- 0 eth
send(tx.value / 3, contract.storage[0])
send(tx.value / 3, contract.storage[1])
send(tx.value / 3, contract.storage[2])
[ALICE, BOB, CHARLIE]
4096ad65:
- 77 eth

Development Environment

- Remix IDE: Browser-based IDE for developing Ethereum contracts.
  - https://remix.ethereum.org/
  - No setup, ready-to-use, nice code editor.
  - Support debug, deploy, testing, code checking, etc., all in browser.
- Truffle: Ethereum development framework (console-based)
  - https://github.com/trufflesuite/truffle
  - Command line tools, powerful, suitable for advanced users.

Smart Contract Development Demo

References

- Solidity Document, https://solidity.readthedocs.io/, where some sample codes in my slides are from.
- Ethereum Developer Resources, https://ethereum.org/developers/, recources listed on Ethereum official website.
- Learn to Code Blockchain DApps By Building Simple Games, https://cryptozombies.io/, strongly recommended as a start point to learn Solidity coding.
- The Ethernaut - Smart Contract Wargame, https://ethernaut.openzeppelin.com/, strongly recommended if you want to learn more about smart contract security.

Q&A

Topic for Next Week: Deployment & Security Considerations