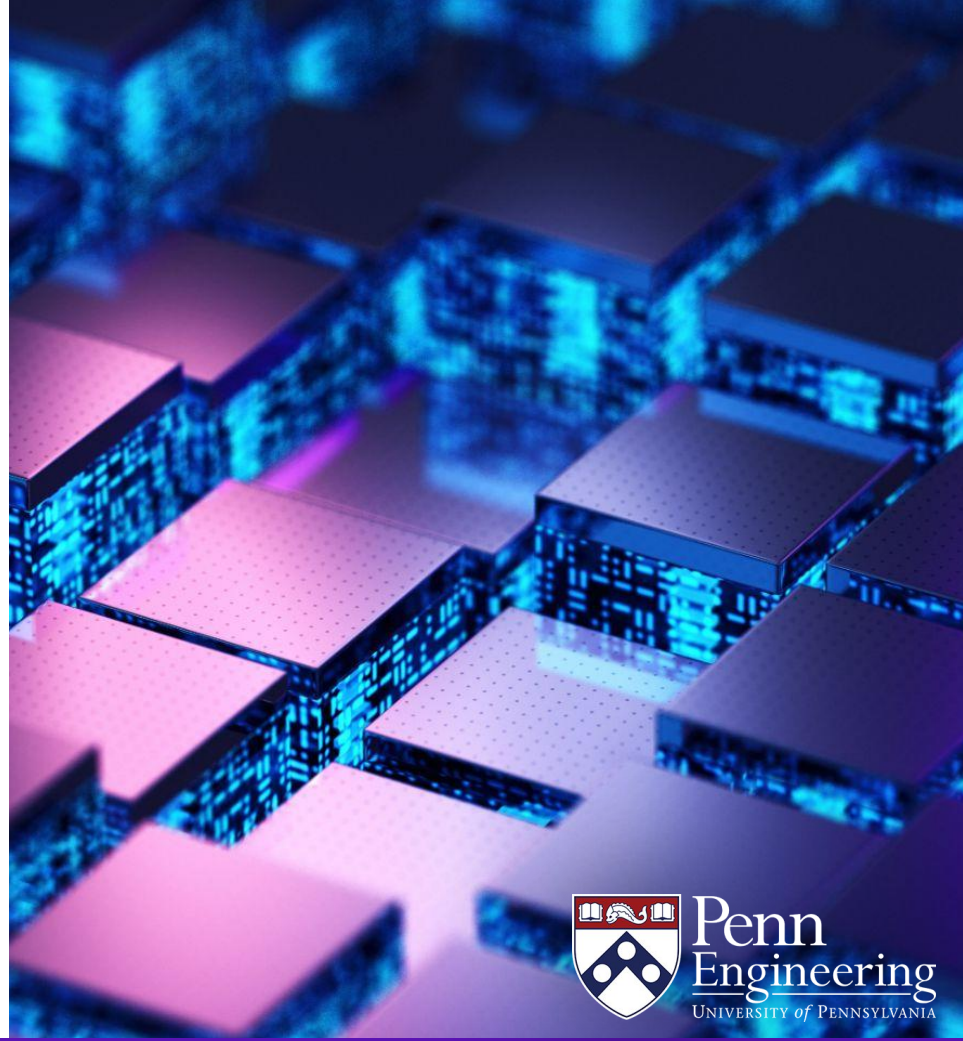
EAS 5830: BLOCKCHAINS

Solidity

Professor Brett Hemenway Falk



Solidity

- The EVM executes byte-code
- Solidity is the most popular language for compiling to the EVM
  - Vyper is the second
- Solidity contracts are much like classes in object-oriented programming languages such as Javascript

```
pragma solidity ^0.8.17;

contract Coin {
    address public minter;
    mapping(address => uint) public balances;

    constructor() { //Only called at contract creation
        minter = msg.sender;
    }

    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);  // Can only be called by the contract creator
        balances[receiver] += amount;
    }

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

Contracts are like classes in object-oriented languages

```
pragma solidity ^0.8.17;

contract Coin {
    address public minter;
    mapping(address => uint) public balances;

    constructor() { //Only called at contract creation
        minter = msg.sender;
    }

    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);  // Can only be called by the contract creator
        balances[receiver] += amount;
    }

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

Two variables:
minter
balances

Value Types

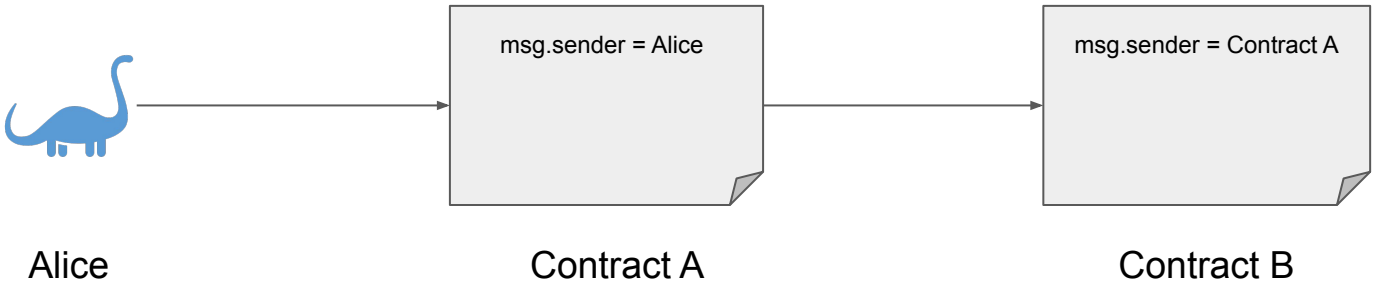
- **Address:** Address hold a 20-byte value representing an Ethereum address.
- **Uint:** Unsigned 256-bit Integer
- Mapping: Like a python dictionary.
  - mapping(address => uint)
    - keys are of type address
    - values are of type uint

```
pragma solidity ^0.8.17;

contract Coin {
    address public minter;
    mapping(address => uint) public balances;

    constructor() { //Only called at contract creation
        minter = msg.sender;
    }

    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);  // Can only be called by the contract creator
        balances[receiver] += amount;
    }

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

Constructor initializes minter variable to msg.sender

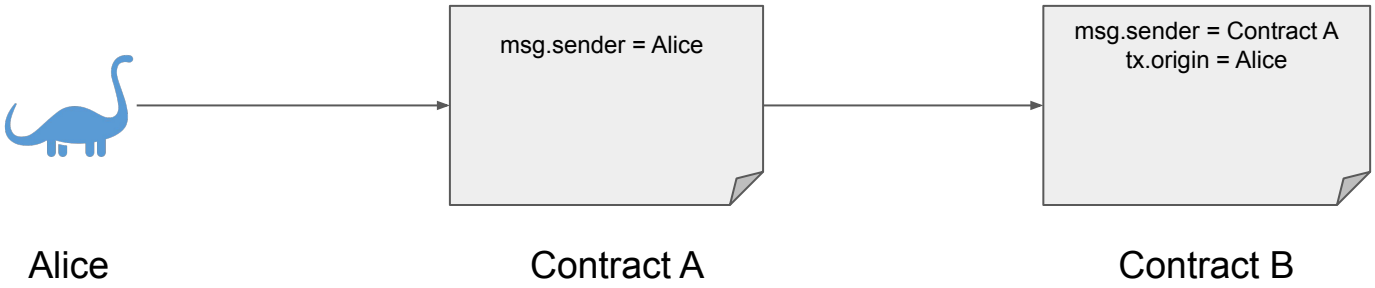
msg.sender

- One key difference between the EVM and a traditional operating system is that every action can be attributed to an identity (address)
- In solidity msg.sender is a variable that holds the address of the entity that called the contract



msg.sender

- One key difference between the EVM and a traditional operating system is that every action can be attributed to an identity (address)
- In solidity msg.sender is a variable that holds the address of the entity that called the contract
- tx.origin holds the originator of the transaction



```
42    modifier callerIsUser() {
43      require(tx.origin == msg.sender, "The caller is another contract");
44      _;
45    }
```

```
pragma solidity ^0.8.17;

contract Coin {
    address public minter;
    mapping(address => uint) public balances;

    constructor() { //Only called at contract creation
        minter = msg.sender;
    }

    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);  // Can only be called by the contract creator
        balances[receiver] += amount;
    }

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

send function decrements sender balance, increments receiver balance

```
pragma solidity ^0.8.17;

contract Coin {
    address public minter;
    mapping(address => uint) public balances;

    constructor() { //Only called at contract creation
        minter = msg.sender;
    }

    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);  // Can only be called by the contract creator
        balances[receiver] += amount;
    }

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

mint function increments receiver balance

```
pragma solidity ^0.8.17;

contract Coin {
    address public minter;
    mapping(address => uint) public balances;

    constructor() { //Only called at contract creation
        minter = msg.sender;
    }

    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);  // Can only be called by the contract creator
        balances[receiver] += amount;
    }

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

Function Visibility

Function Visibility

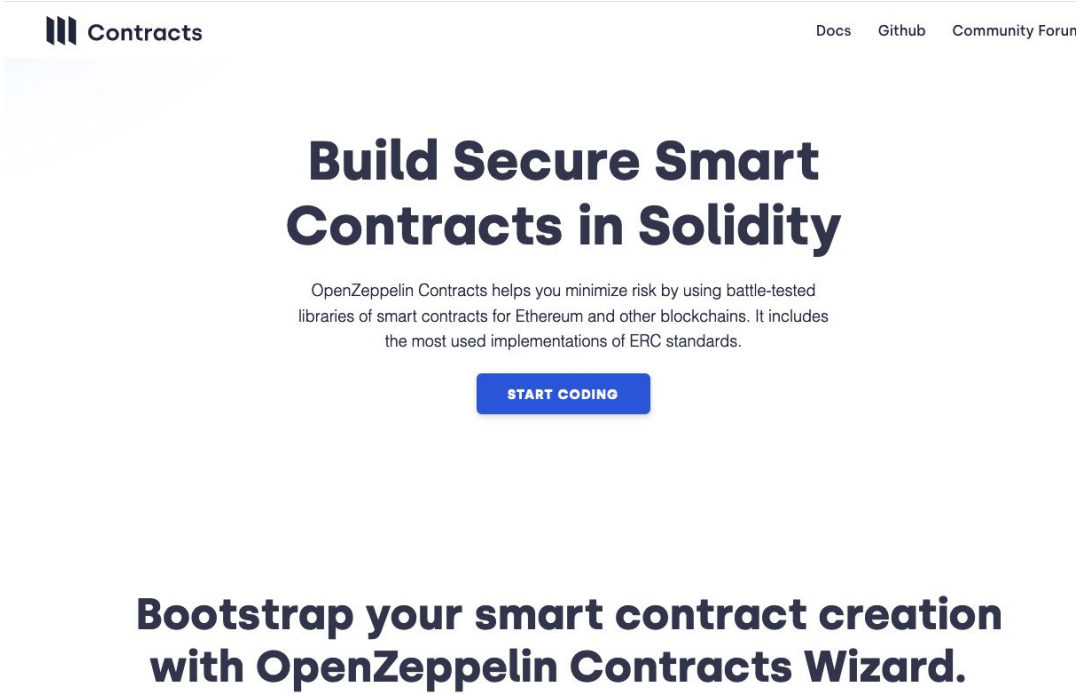
Solidity has four types of visibilities for both functions and variables:

- **Public:** Can be called internally or through messages
- **External:** Can be called from other contracts and transactions. They cannot be called internally, except with "this.functionName()"
- **Private:** Only available to the current contract and **not** derived contracts
  - Private functions are **not** secret (their state can be read publicly)
- **Internal:** Can only be accessed internally (current contract or derived)

Function Visibility

	This contract	Derived Contracts	Other Contract EOAs
Public			
External			
Private			
Internal			

OpenZeppelin



```solidity
// contracts/GLDToken.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract GLDToken is ERC20 {
    constructor(uint256 initialSupply) ERC20("Gold", "GLD") {
        _mint(msg.sender, initialSupply);
    }
}
```

```sol
// contracts/GLDToken.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract GLDToken is ERC20 {
    constructor(uint256 initialSupply) ERC20("Gold", "GLD") {
        _mint(msg.sender, initialSupply);
    }
}
```


FOUNDRY