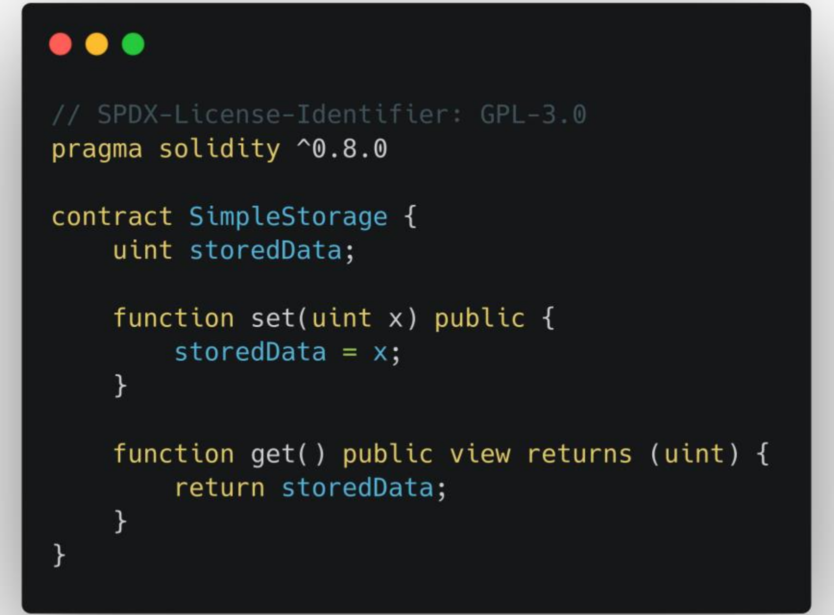
BBS: Smart Contracts Workshop #2

bristol.ac.uk

Solidity

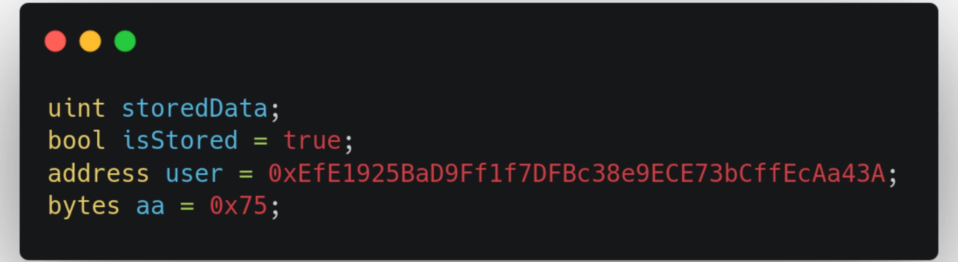
- Licence
- Pragma
- Contracts
- Types
- Variables
  - public, internal or private
- Functions



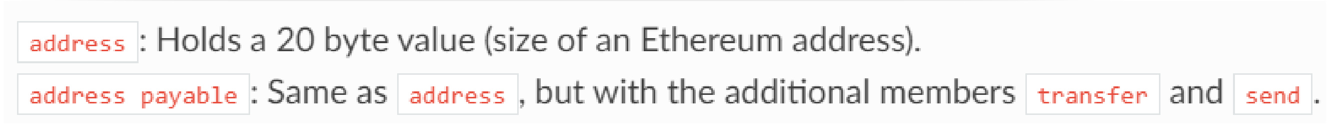
```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.0

contract SimpleStorage {
    uint storedData;

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

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

Solidity

- Basic data types:
  - int,
  - uint,
  - enum,
  - bytes etc
- Blockchain specific data types:
  - address (20 bytes)



```
uint storedData;
bool isStored = true;
address user = 0xEfE1925BaD9Ff1f7DFBc38e9ECE73bCffEcAa43A;
bytes aa = 0x75;
```



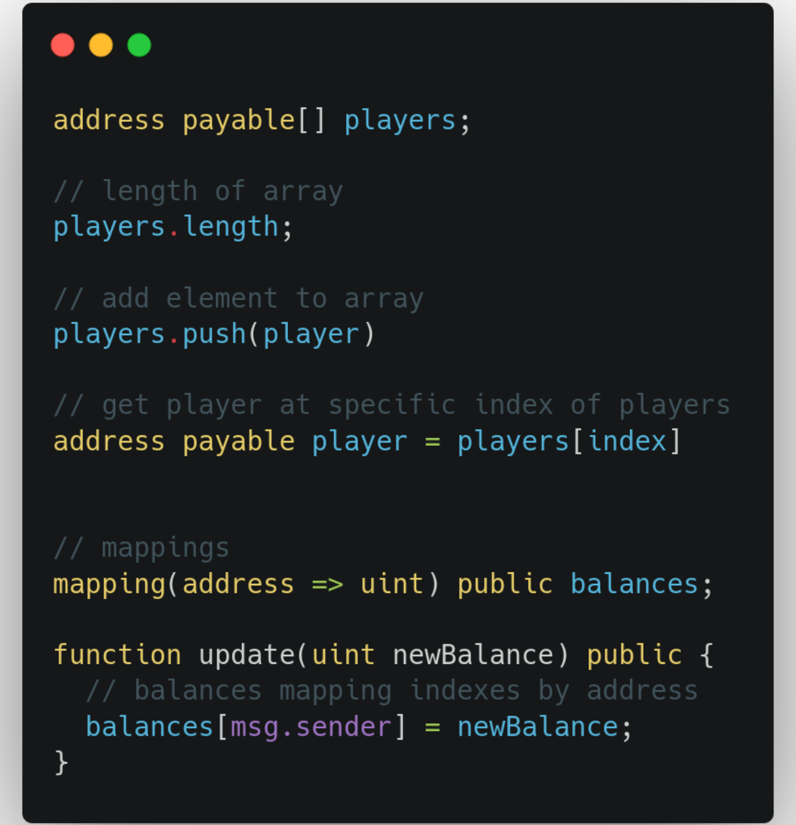
`address`: Holds a 20 byte value (size of an Ethereum address).

`address payable`: Same as `address`, but with the additional members `transfer` and `send`.

Solidity

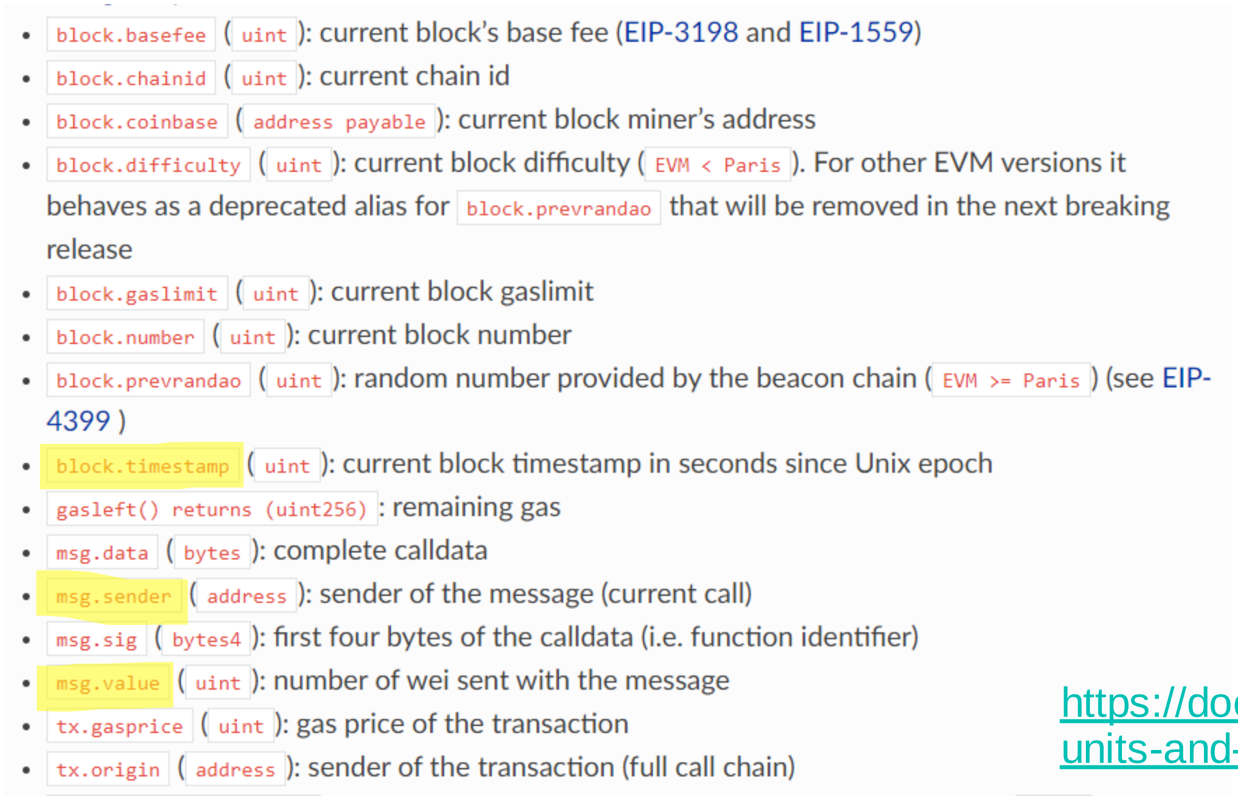
Data structures:

- Addresses
- Arrays
- Structs
- Mappings
- Data Location:
  - Calldata,
  - Memory or
  - Storage



```
address payable[] players;

// length of array
players.length;

// add element to array
players.push(player)

// get player at specific index of players
address payable player = players[index]


// mappings
mapping(address => uint) public balances;

function update(uint newBalance) public {
  // balances mapping indexes by address
  balances[msg.sender] = newBalance;
}
```

Solidity: Globally available variables/functions

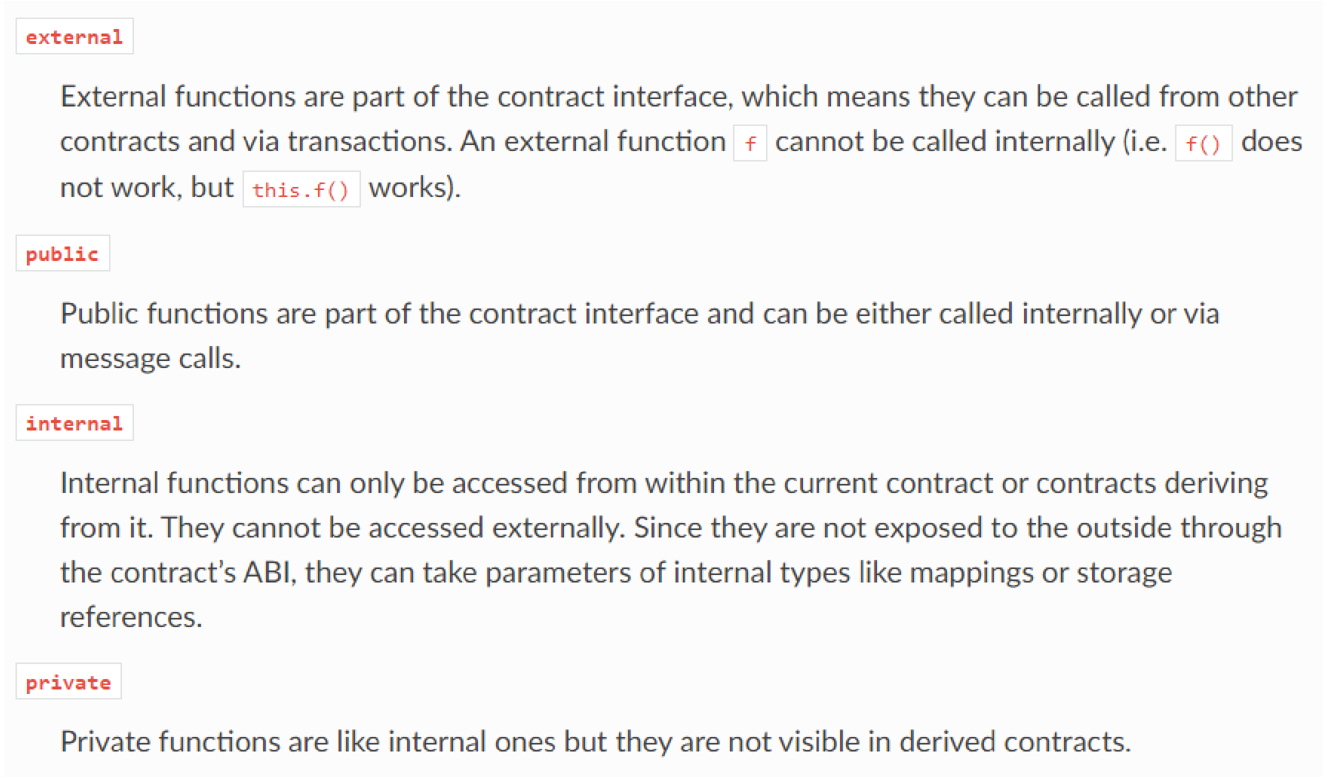


- `block.basefee` (`uint`): current block's base fee (EIP-3198 and EIP-1559)
- `block.chainid` (`uint`): current chain id
- `block.coinbase` (`address payable`): current block miner's address
- `block.difficulty` (`uint`): current block difficulty (`EVM < Paris`). For other EVM versions it behaves as a deprecated alias for `block.prevrandao` that will be removed in the next breaking release
- `block.gaslimit` (`uint`): current block gaslimit
- `block.number` (`uint`): current block number
- `block.prevrandao` (`uint`): random number provided by the beacon chain (`EVM >= Paris`) (see EIP-4399)
- `block.timestamp` (`uint`): current block timestamp in seconds since Unix epoch
- `gasleft() returns (uint256)` : remaining gas
- `msg.data` (`bytes`): complete calldata
- `msg.sender` (`address`): 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
- `tx.gasprice` (`uint`): gas price of the transaction
- `tx.origin` (`address`): sender of the transaction (full call chain)

https://docs.soliditylang.org/en/latest/units-and-global-variables.html

Solidity: Functions

- Visibility:
  - public
  - external
  - internal
  - private
- State Mutability:
  - pure
  - view
  - payable



`external`

External functions are part of the contract interface, which means they can be called from other contracts and via transactions. An external function `f` cannot be called internally (i.e. `f()` does not work, but `this.f()` works).

`public`

Public functions are part of the contract interface and can be either called internally or via message calls.

`internal`

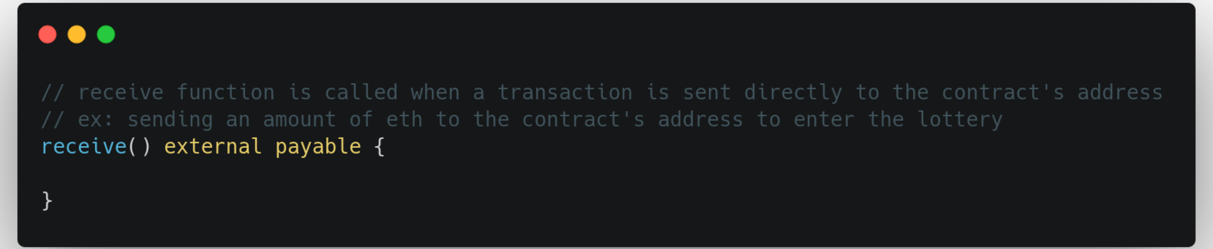
Internal functions can only be accessed from within the current contract or contracts deriving from it. They cannot be accessed externally. Since they are not exposed to the outside through the contract's ABI, they can take parameters of internal types like mappings or storage references.

`private`

Private functions are like internal ones but they are not visible in derived contracts.

Solidity Built-in Functions

- receive() Function
  - This function is called when eth is sent directly to the contract's address without calling a specific smart contract function
  - Used as an extension to enterLottery() function so that players can send eth to the contract and automatically enter the lottery.



```
// receive function is called when a transaction is sent directly to the contract's address
// ex: sending an amount of eth to the contract's address to enter the lottery
receive() external payable {

}
```

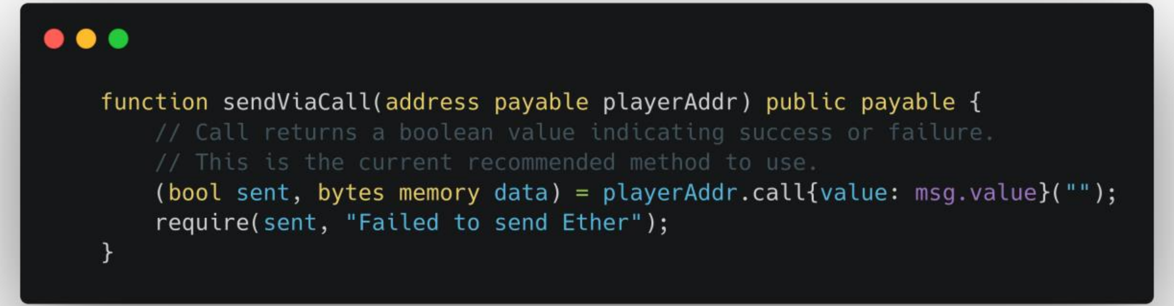
Solidity: Sending Eth

- Transfer, send, call
  - Call is recommended
- Withdrawal pattern

`transfer` (2300 gas, throws error)

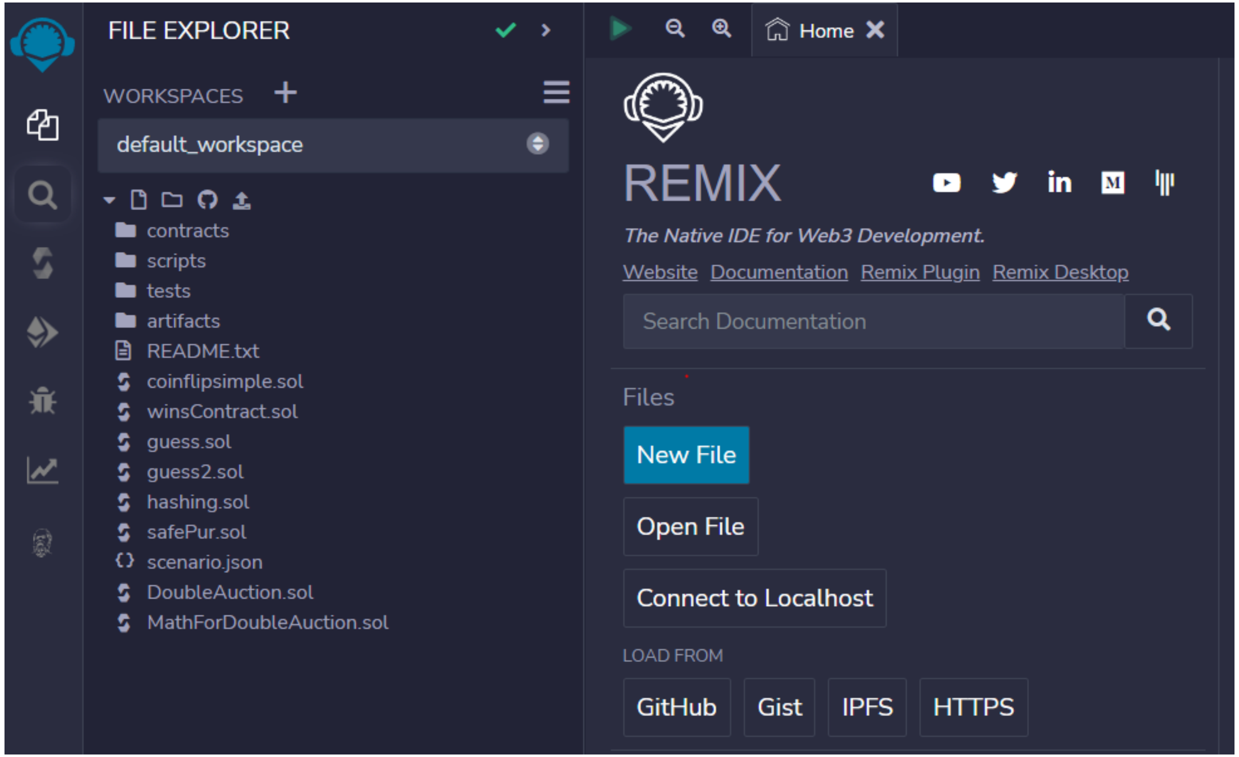
`send` (2300 gas, returns bool)

`call` (forward all gas or set gas, returns bool)



```
function sendViaCall(address payable playerAddr) public payable {
    // Call returns a boolean value indicating success or failure.
    // This is the current recommended method to use.
    (bool sent, bytes memory data) = playerAddr.call{value: msg.value}("");
    require(sent, "Failed to send Ether");
}
```

Remix IDE



Remix IDE is an open-source web application for smart contract development

Website: remix.ethereum.org

Worksheet and Skeleton code

- https://github.com/Bristol-Blockchain-Society/lottery-workshop