Lesson 4:

⇒ var and var-**hoisting** has been removed from solidity version 5.0

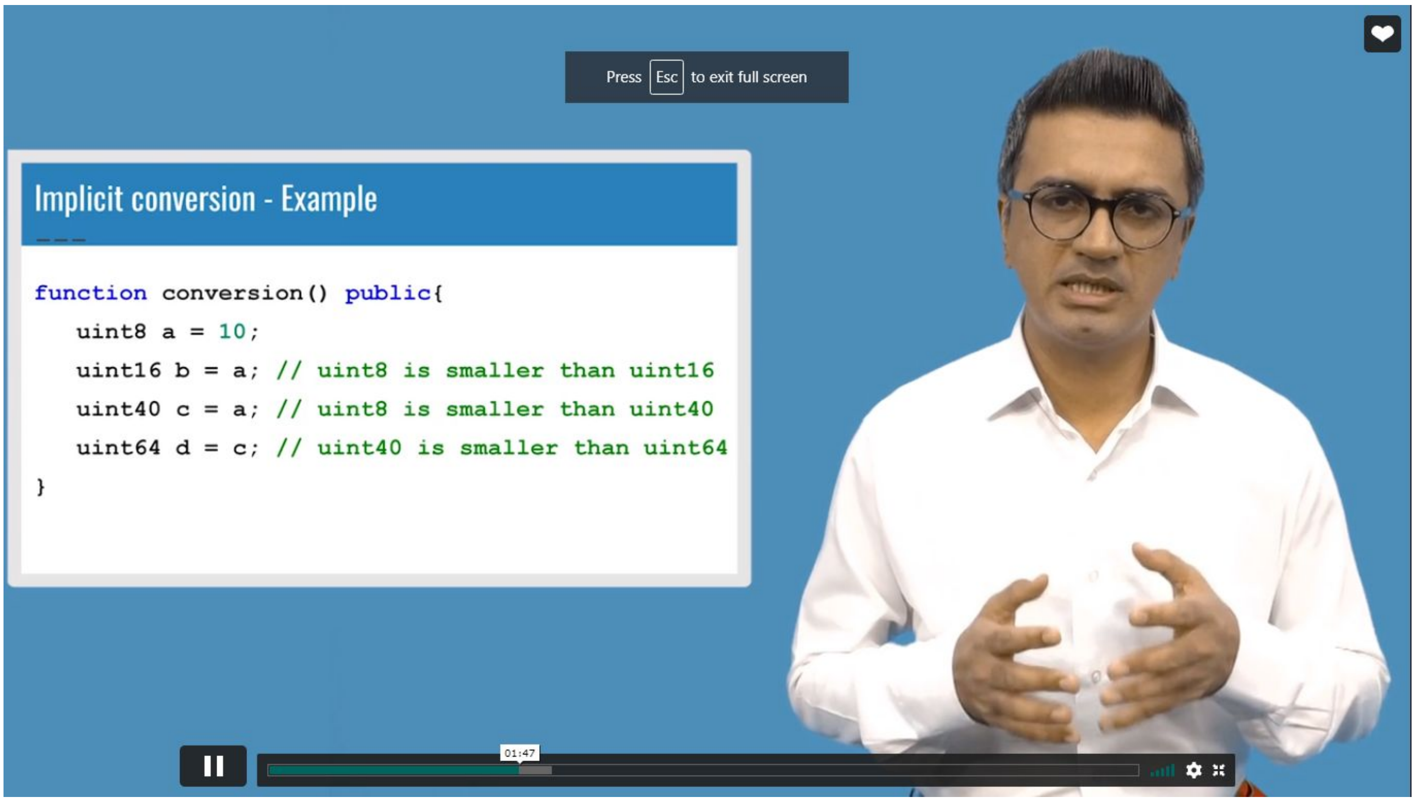
⇒ Solidity is a static type language

Type Conversion:

1. Implicit Type Casting
2. Explicit Type Casting

Implicit Conversion:
⇒ mean storing a single data type to a larger data type (it will be automatic there is no need for an operator)
e.g.
 uint8 a = 12; // range from 0-255
uint16 b = a; // range from 0-65535

⇒ Its type safe conversion

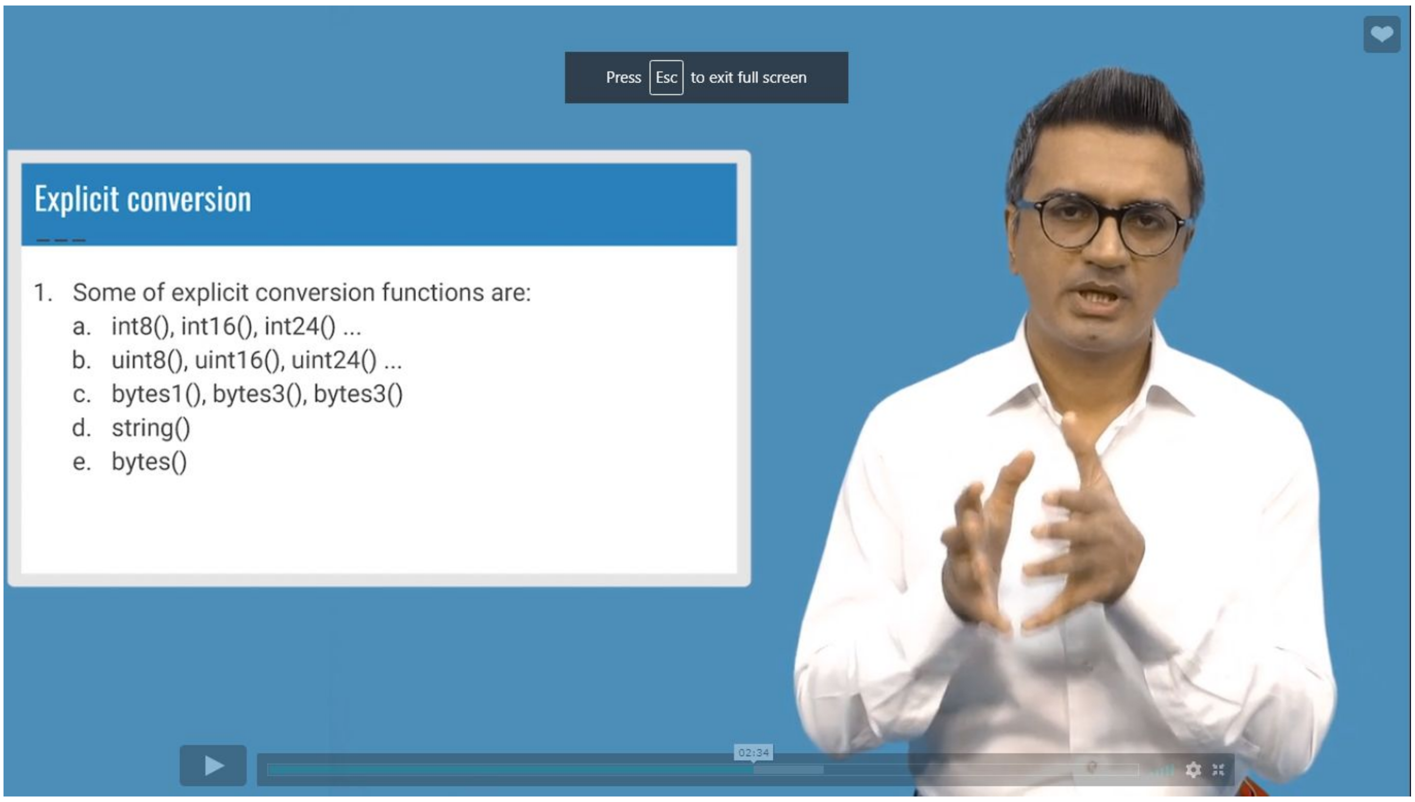


Implicit conversion - Example

```
function conversion() public{
    uint8 a = 10;
    uint16 b = a; // uint8 is smaller than uint16
    uint40 c = a; // uint8 is smaller than uint40
    uint64 d = c; // uint40 is smaller than uint64
}
```

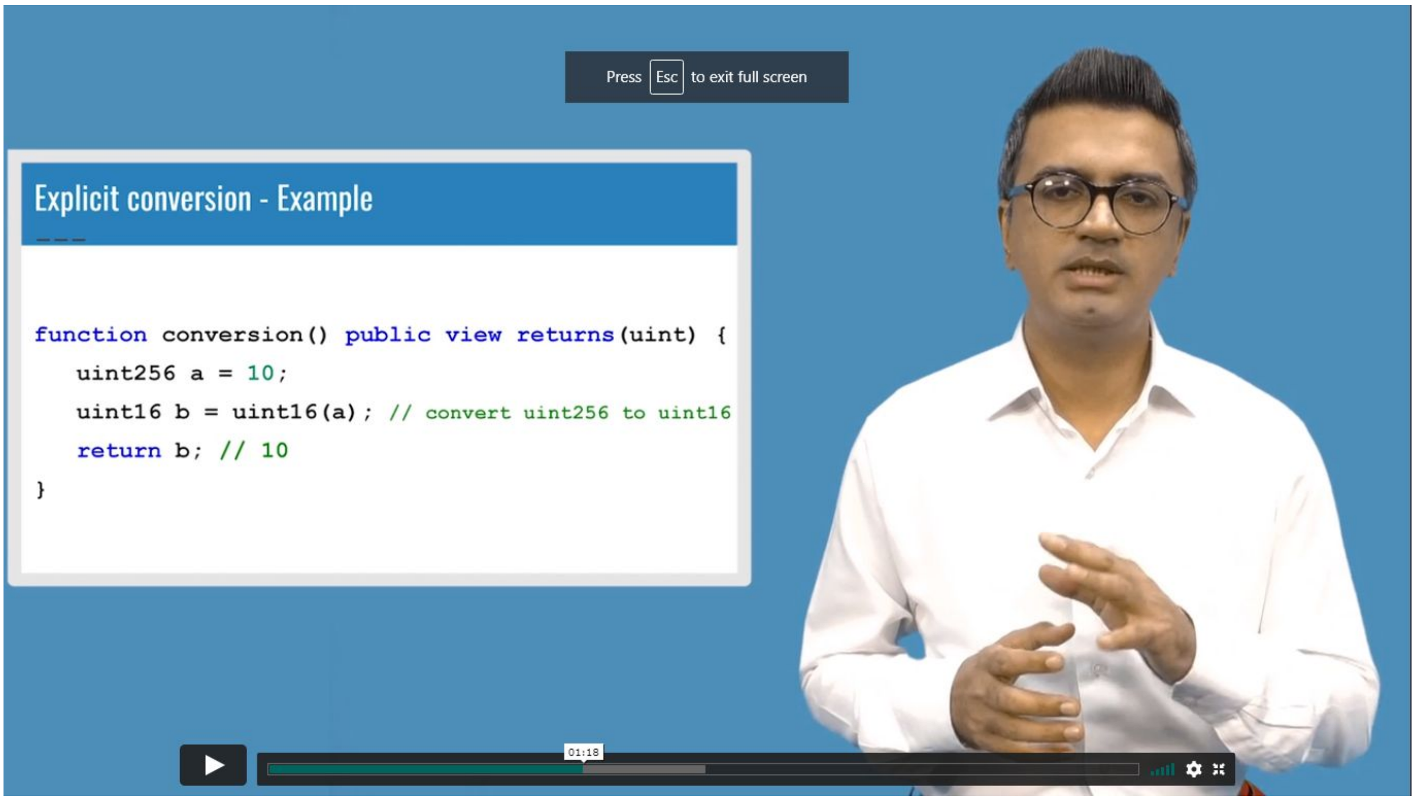
Explicit Conversion:

⇒ storing larger data type to small data type (forcefully)



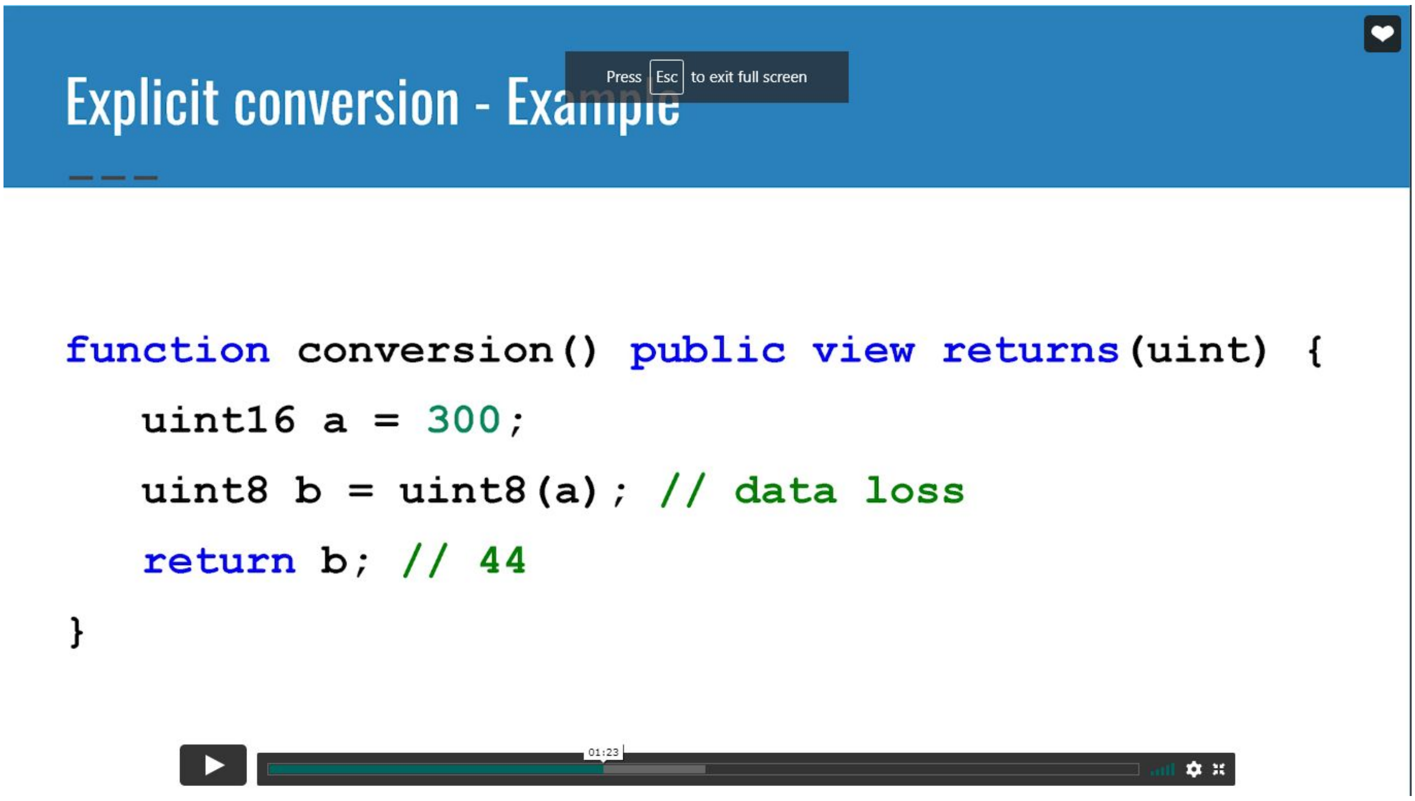
Explicit conversion

1. Some of explicit conversion functions are:
   a. int8(), int16(), int24() ...
   b. uint8(), uint16(), uint24() ...
   c. bytes1(), bytes3(), bytes3()
   d. string()
   e. bytes()



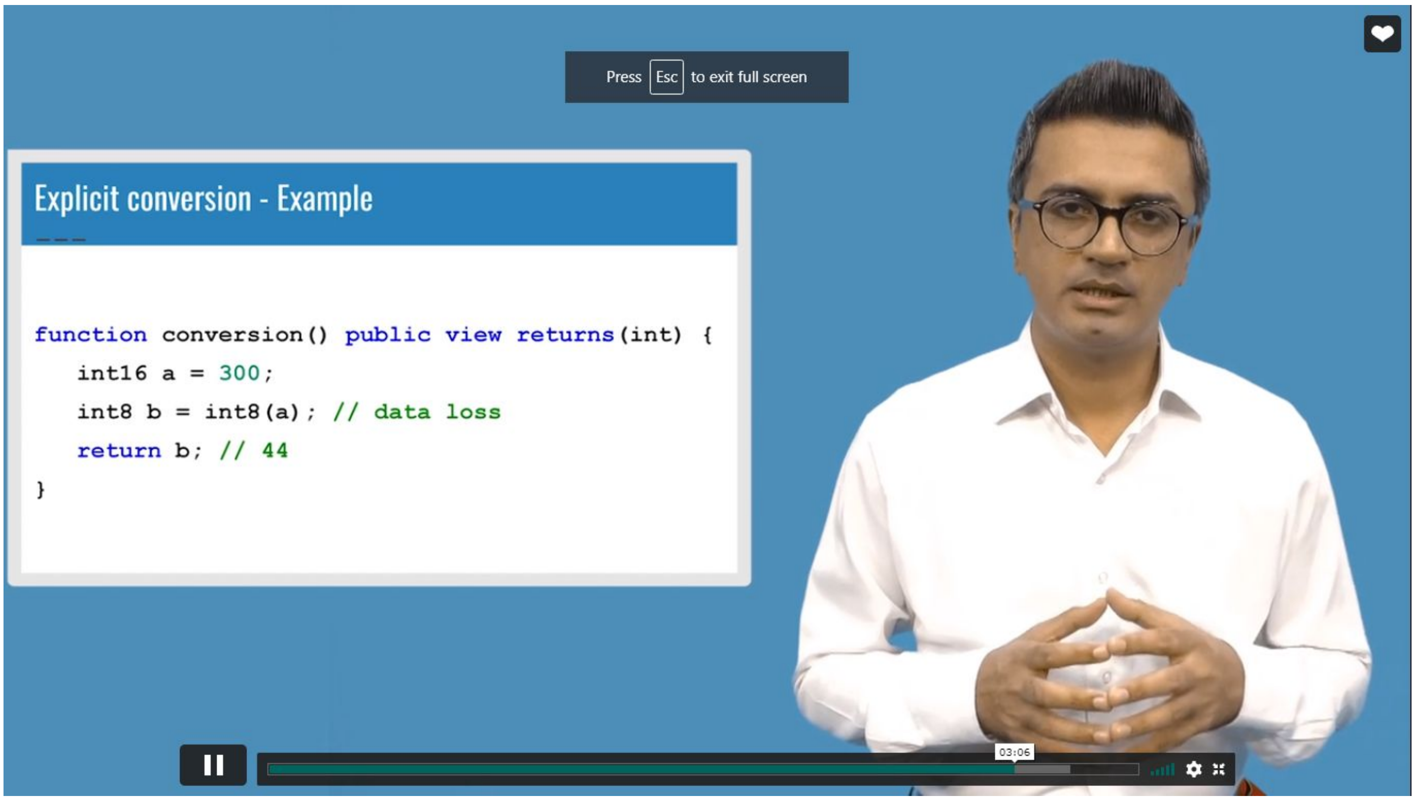
Explicit conversion - Example

```
function conversion() public view returns(uint) {
   uint256 a = 10;
   uint16 b = uint16(a); // convert uint256 to uint16
   return b; // 10
}
```



Explicit conversion - Example

```
function conversion() public view returns(uint) {
   uint16 a = 300;
   uint8 b = uint8(a); // data loss
   return b; // 44
}
```



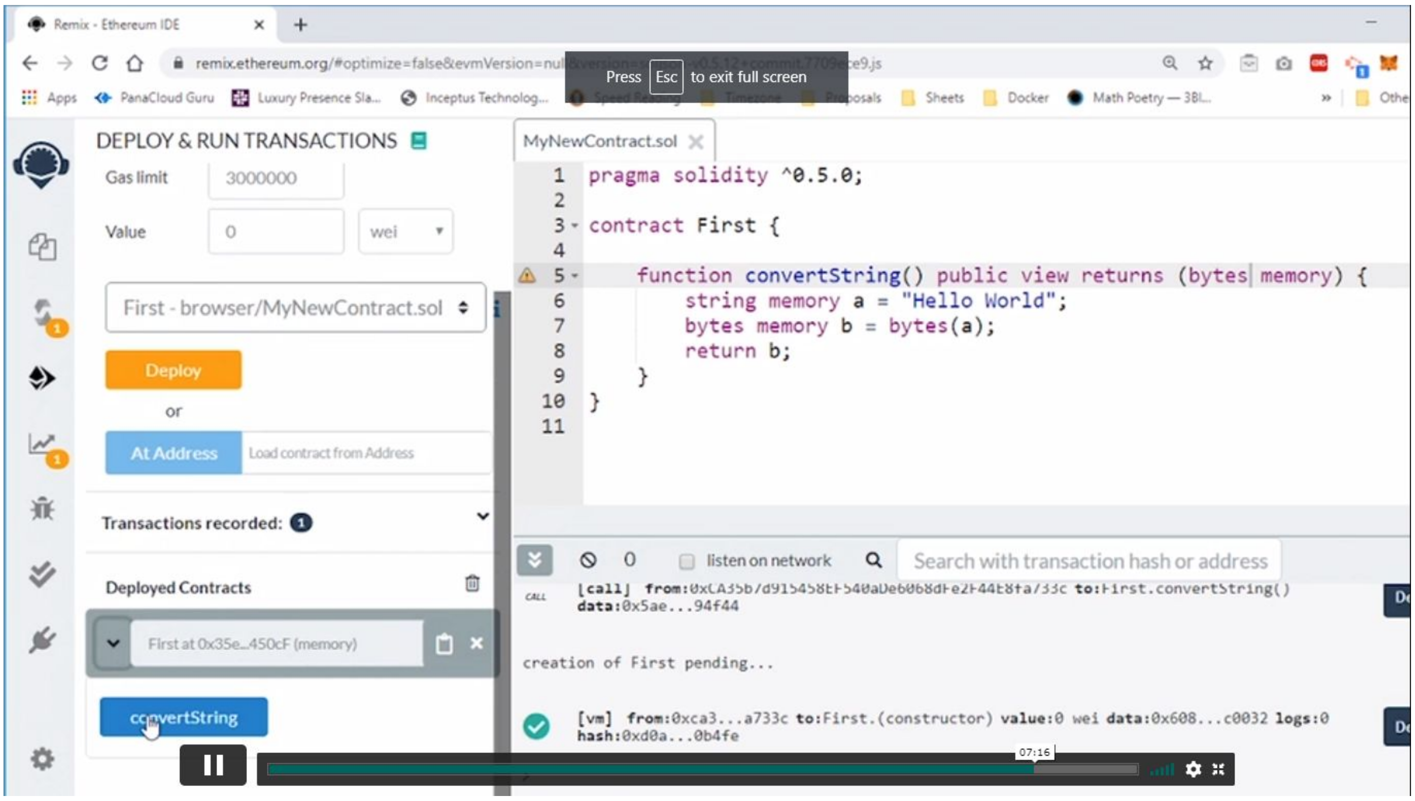
Explicit conversion - Example

```
function conversion() public view returns(int) {
   int16 a = 300;
   int8 b = int8(a); // data loss
   return b; // 44
}
```

If data is lost it will return the available bits value

⇒ **Solidity return enum value in uint**

⇒ String to bytes conversion

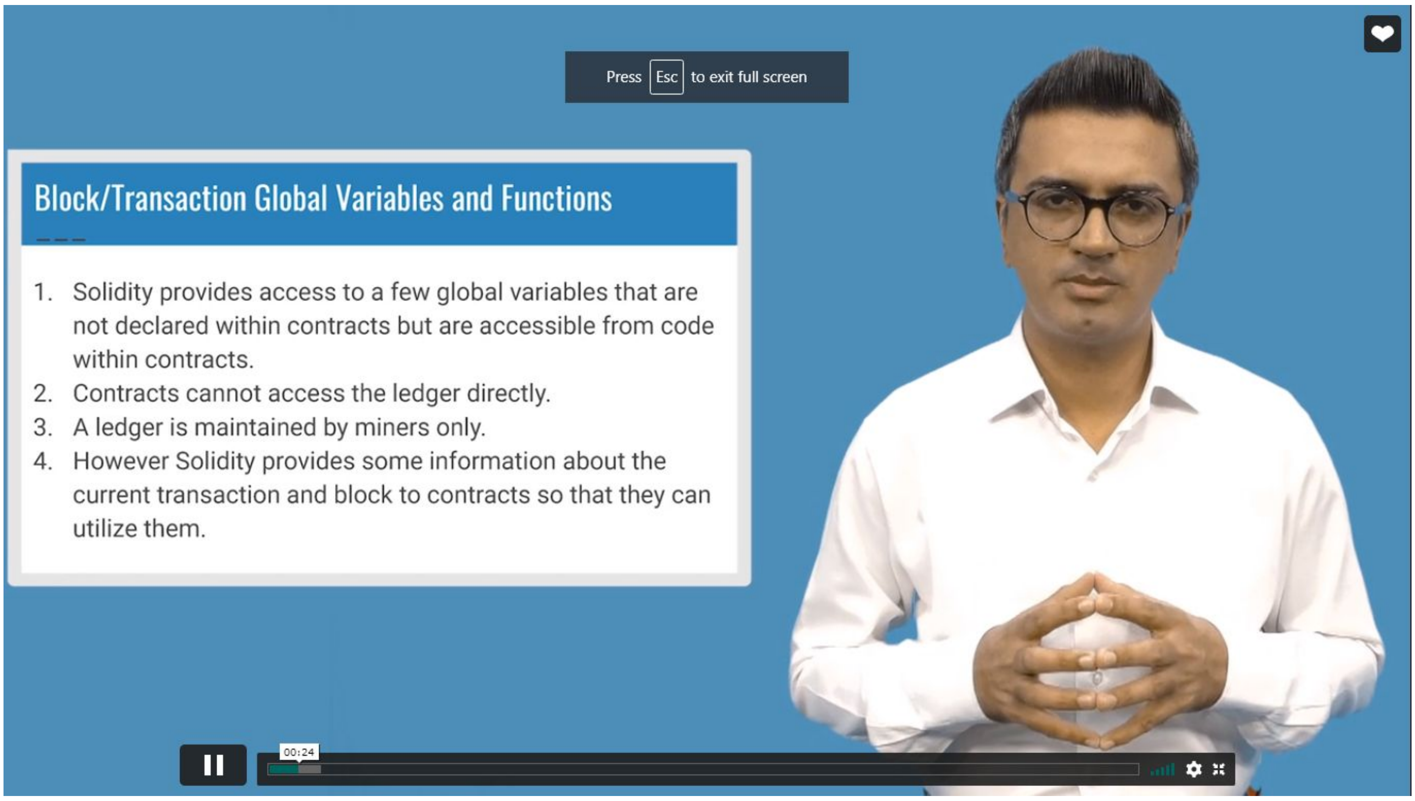

Remix - Ethereum IDE
Press Esc to exit full screen
DEPLOY & RUN TRANSACTIONS
Gas limit 3000000
Value 0 wei
First - browser/MyNewContract.sol
Deploy
or
At Address
Load contract from Address
Transactions recorded: 1
Deployed Contracts
First at 0x35e...450cF (memory)
convertString
MyNewContract.sol
listen on network
Search with transaction hash or address
07:16

```
pragma solidity ^0.5.0;

contract First {

    function convertString() public view returns (bytes memory) {
        string memory a = "Hello World";
        bytes memory b = bytes(a);
        return b;
    }
}
```

```
[call] from:0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c to:First.convertString()
data:0x5ae...94f44

creation of First pending...

[vm] from:0xca3...a733c to:First.(constructor) value:0 wei data:0x608...c0032 logs:0
hash:0xd0a...0b4fe
```

Block Global Variables and Transaction:

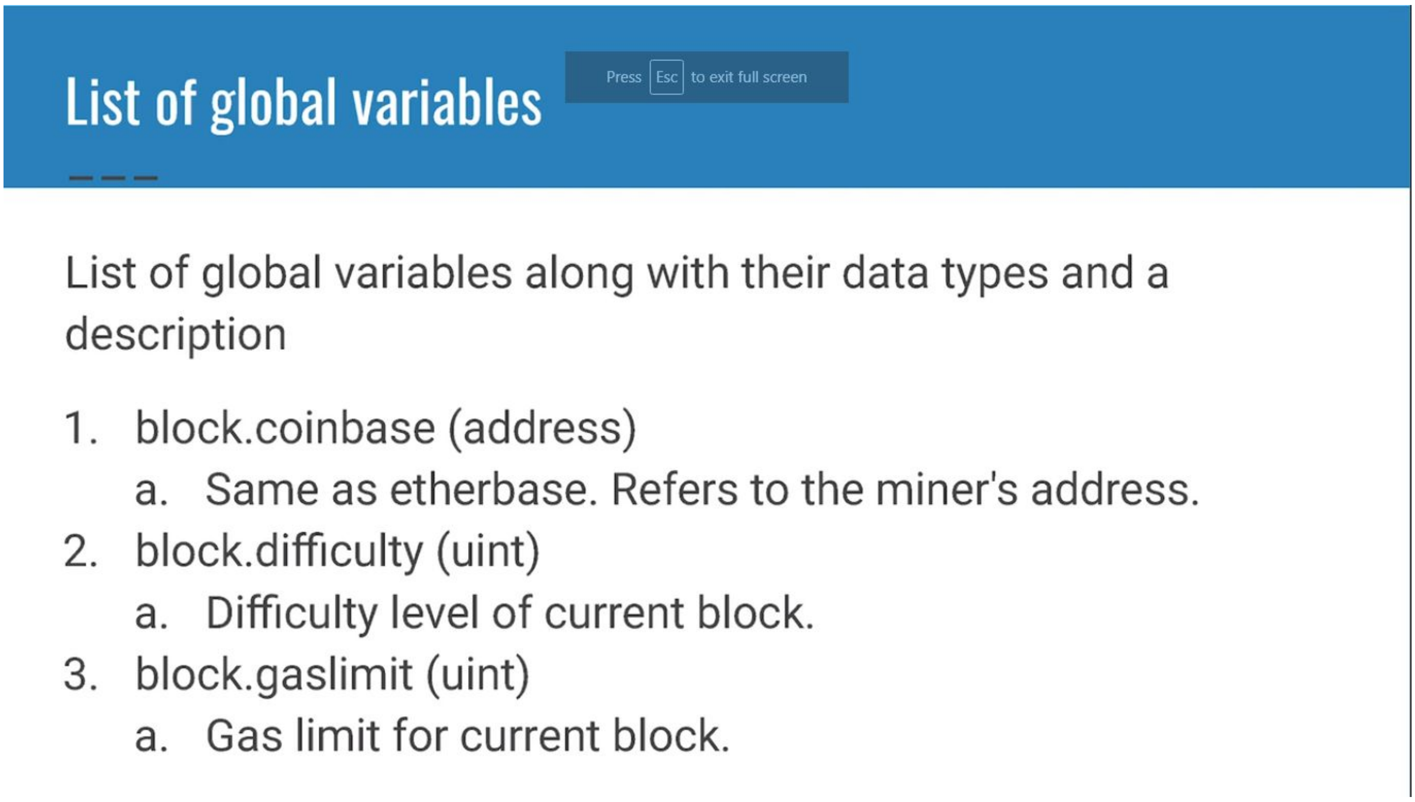
⇒ contract has do not directly access to the ledger



Block/Transaction Global Variables and Functions

1. Solidity provides access to a few global variables that are not declared within contracts but are accessible from code within contracts.
2. Contracts cannot access the ledger directly.
3. A ledger is maintained by miners only.
4. However Solidity provides some information about the current transaction and block to contracts so that they can utilize them.

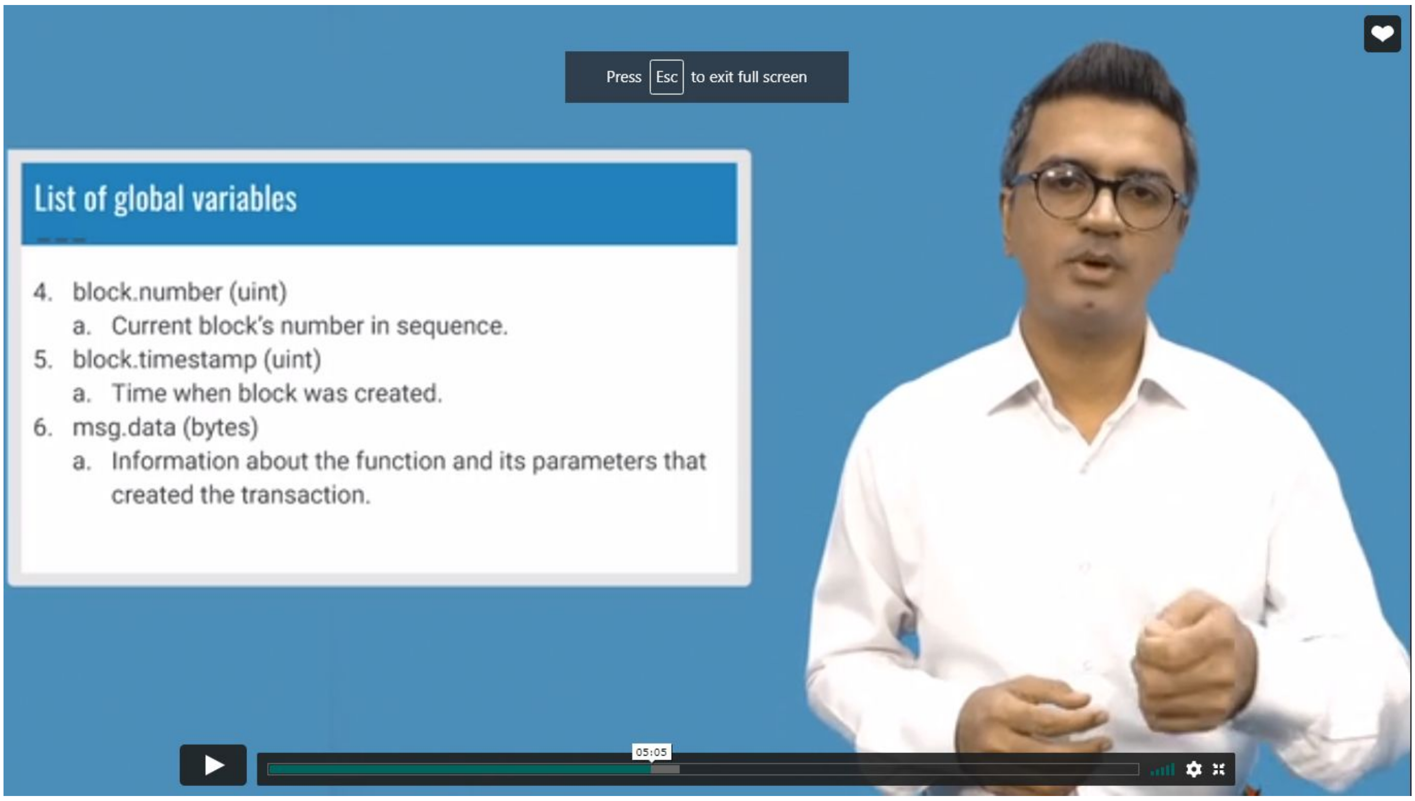
List of Global Variables:



List of global variables

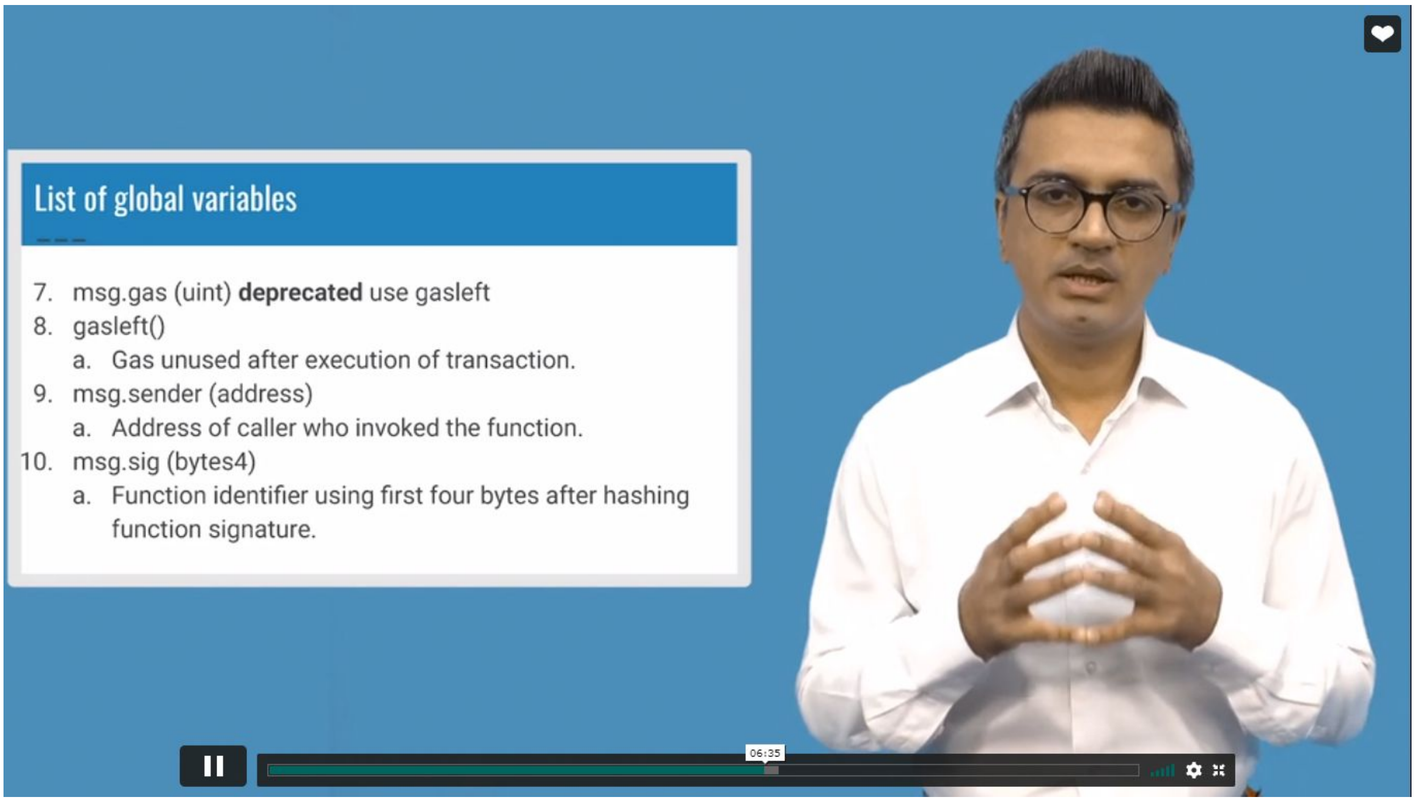
List of global variables along with their data types and a description

1. block.coinbase (address)
   a. Same as etherbase. Refers to the miner's address.
2. block.difficulty (uint)
   a. Difficulty level of current block.
3. block.gaslimit (uint)
   a. Gas limit for current block.



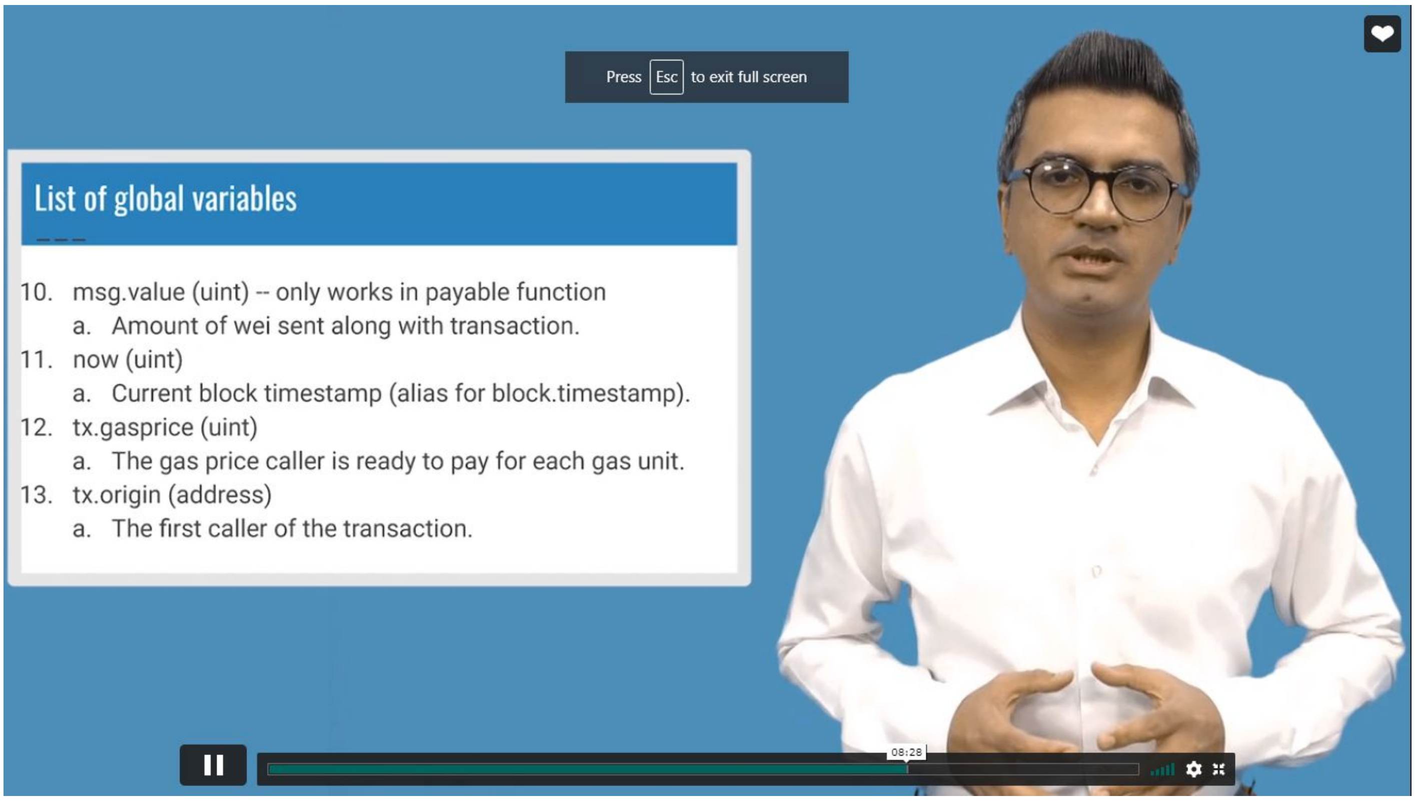
List of global variables

4. block.number (uint)
    a. Current block's number in sequence.
5. block.timestamp (uint)
    a. Time when block was created.
6. msg.data (bytes)
    a. Information about the function and its parameters that created the transaction.



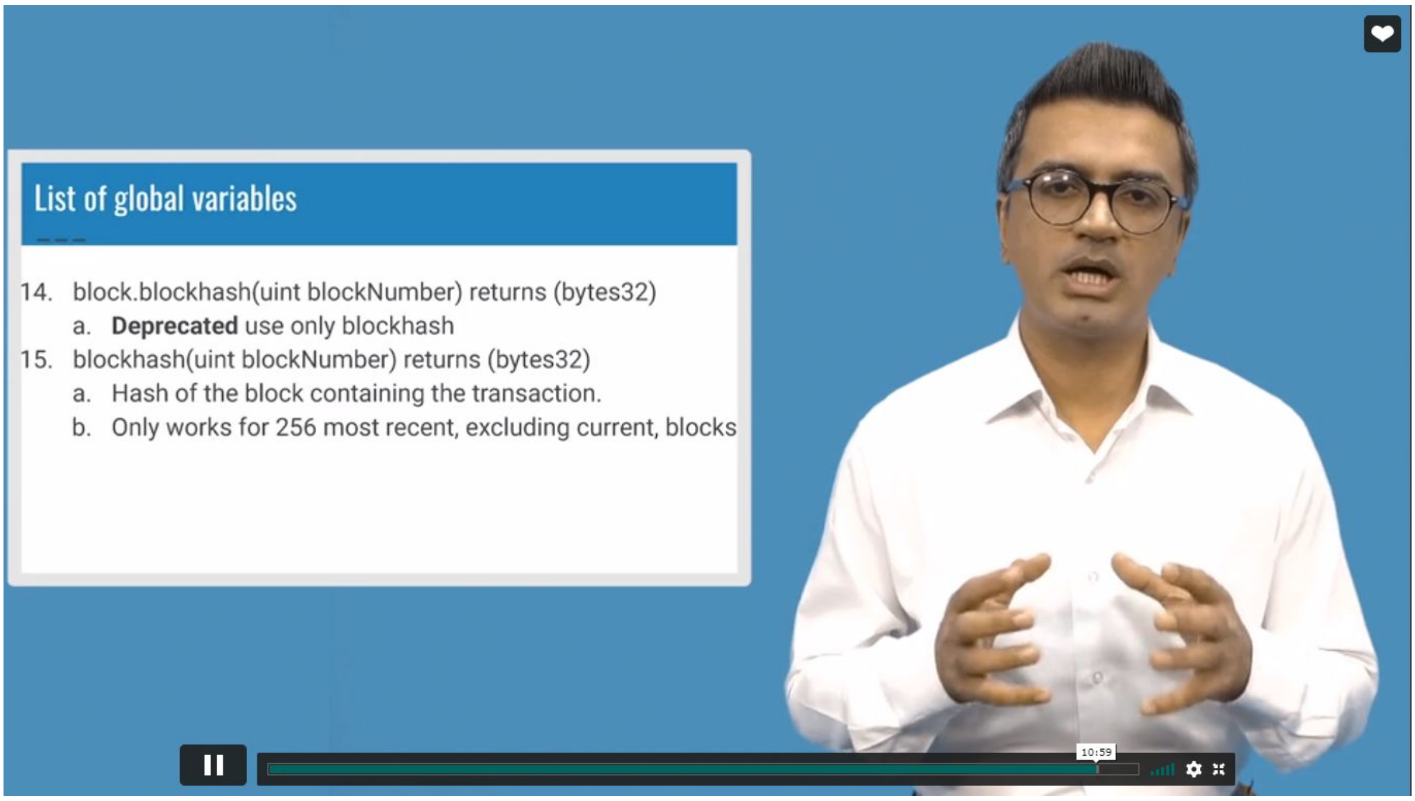
List of global variables

7. msg.gas (uint) **deprecated** use gasleft
8. gasleft()
    a. Gas unused after execution of transaction.
9. msg.sender (address)
    a. Address of caller who invoked the function.
10. msg.sig (bytes4)
    a. Function identifier using first four bytes after hashing function signature.



List of global variables

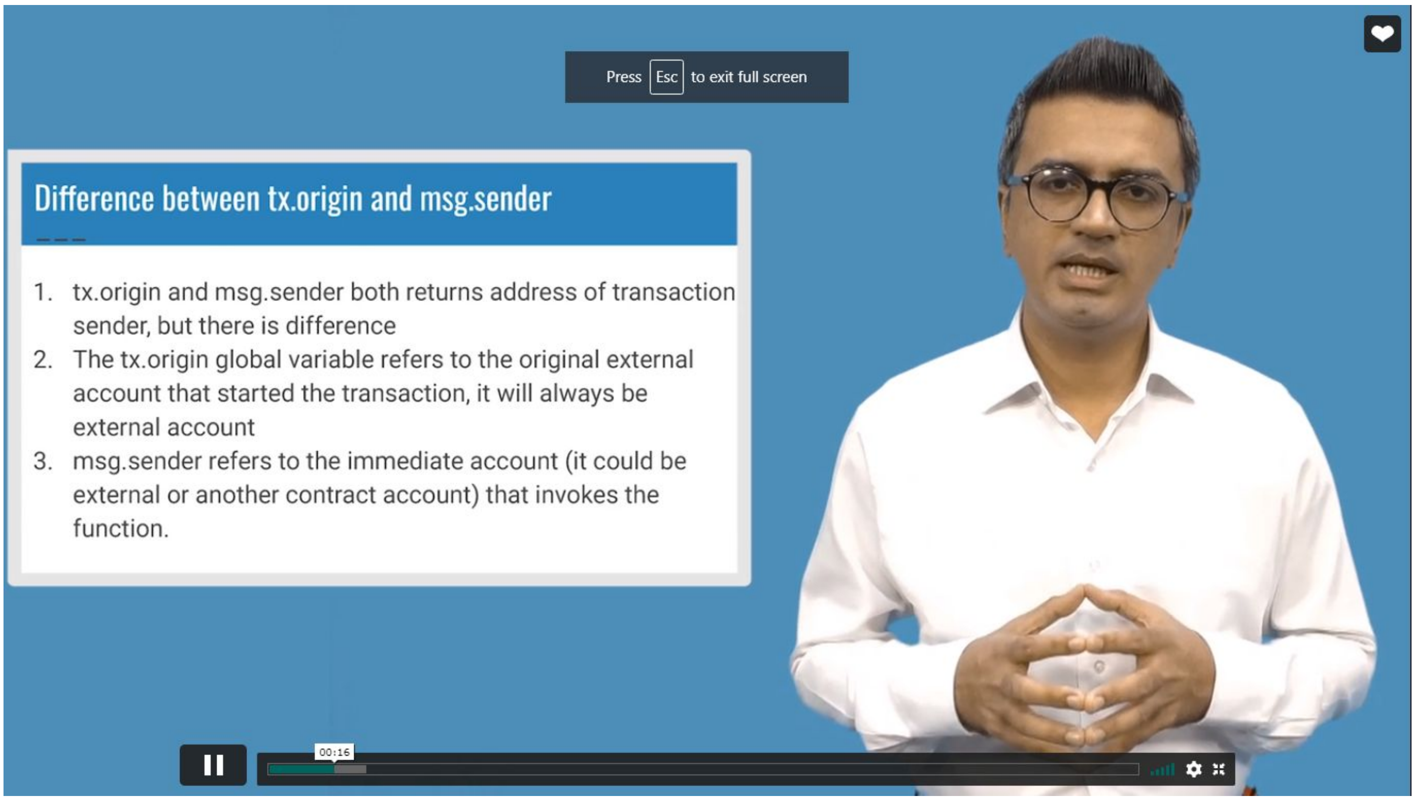
10. msg.value (uint) -- only works in payable function
    a. Amount of wei sent along with transaction.
11. now (uint)
    a. Current block timestamp (alias for block.timestamp).
12. tx.gasprice (uint)
    a. The gas price caller is ready to pay for each gas unit.
13. tx.origin (address)
    a. The first caller of the transaction.



List of global variables

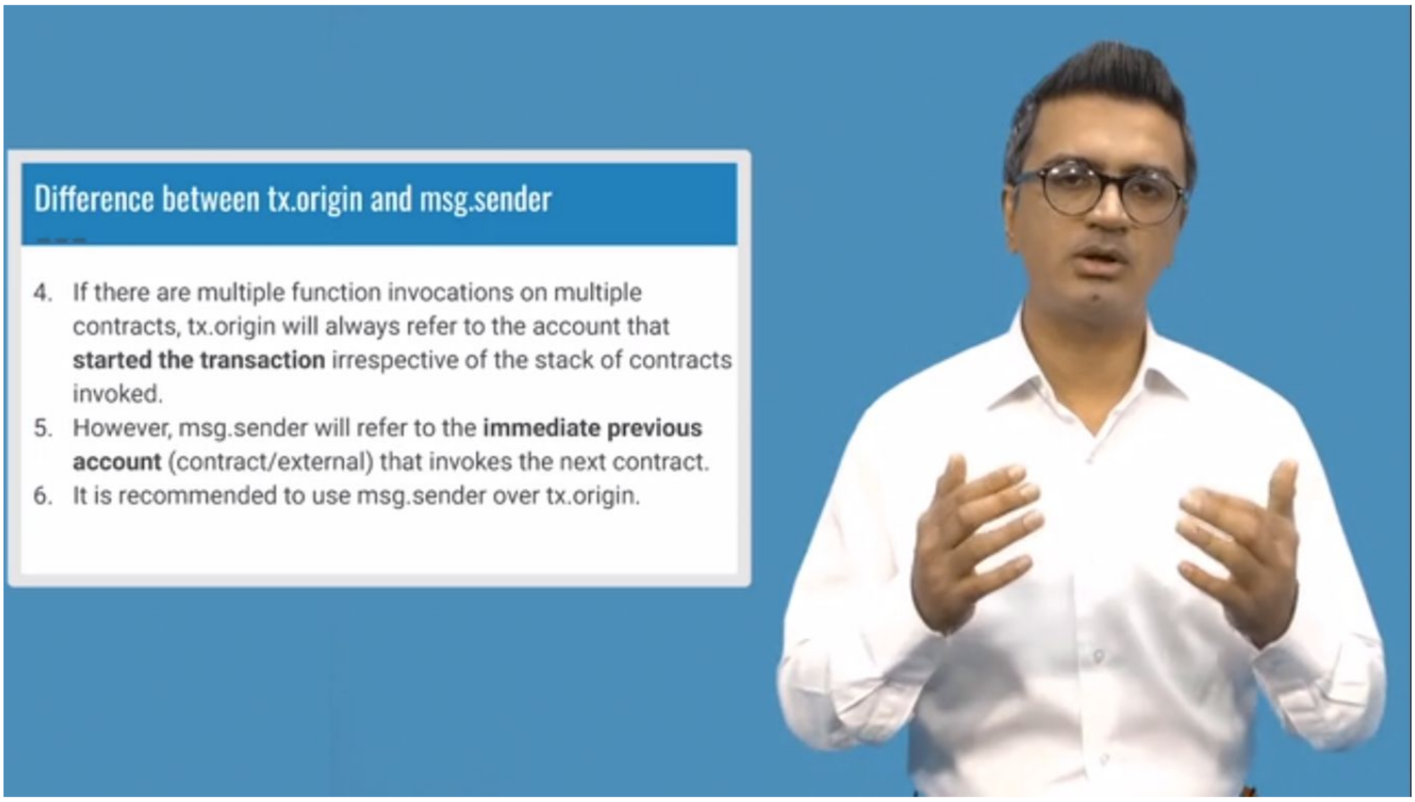
14. block.blockhash(uint blockNumber) returns (bytes32)
    a. **Deprecated** use only blockhash
15. blockhash(uint blockNumber) returns (bytes32)
    a. Hash of the block containing the transaction.
    b. Only works for 256 most recent, excluding current, blocks

Difference between tx.origin and msg.sender



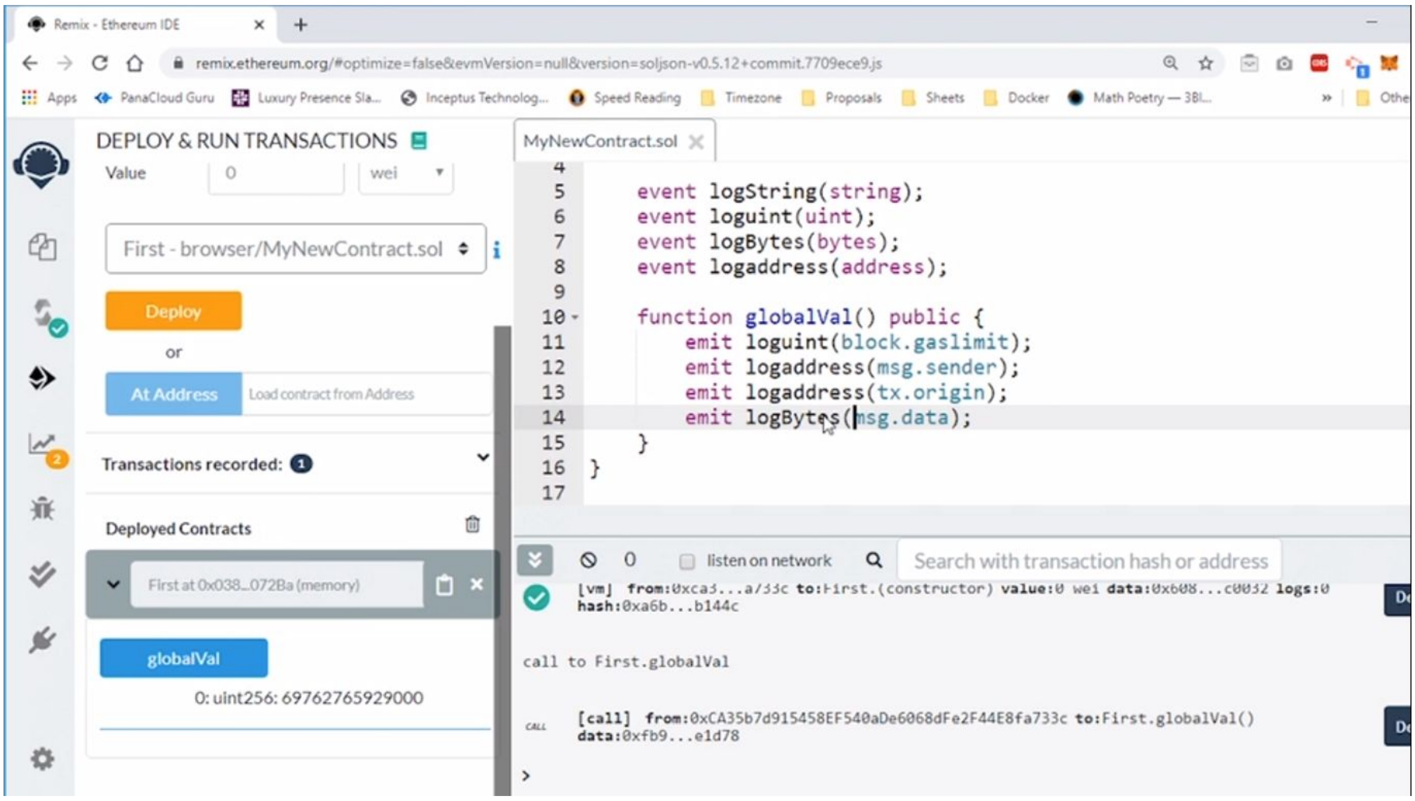
Difference between tx.origin and msg.sender

1. tx.origin and msg.sender both returns address of transaction sender, but there is difference
2. The tx.origin global variable refers to the original external account that started the transaction, it will always be external account
3. msg.sender refers to the immediate account (it could be external or another contract account) that invokes the function.



Difference between tx.origin and msg.sender

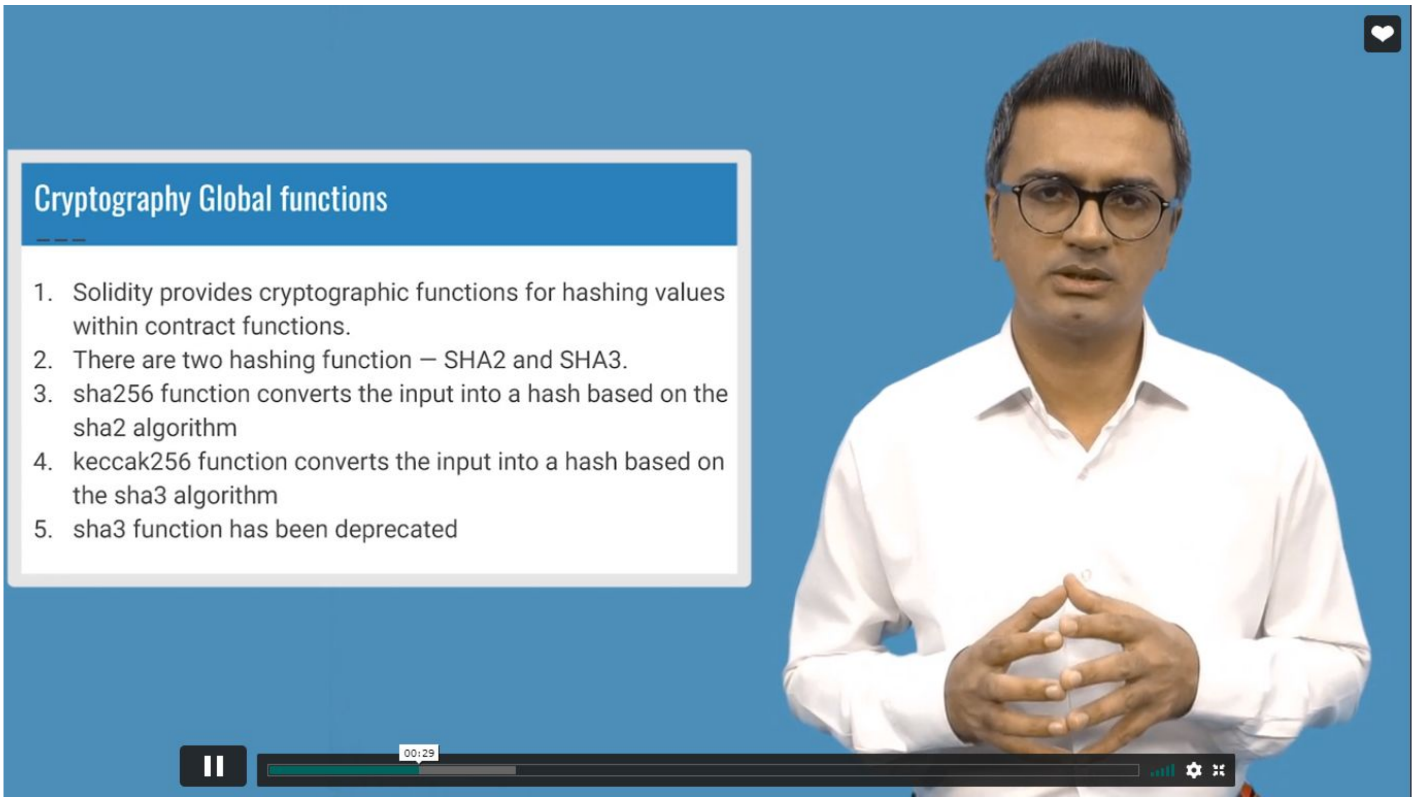
4. If there are multiple function invocations on multiple contracts, tx.origin will always refer to the account that **started the transaction** irrespective of the stack of contracts invoked.
5. However, msg.sender will refer to the **immediate previous account** (contract/external) that invokes the next contract.
6. It is recommended to use msg.sender over tx.origin.


Remix - Ethereum IDE
remix.ethereum.org/#optimize=false&evmVersion=null&version=soljson-v0.5.12+commit.7709ece9.js
DEPLOY & RUN TRANSACTIONS
Value
0
wei
First - browser/MyNewContract.sol
Deploy
or
At Address
Load contract from Address
Transactions recorded: 1
Deployed Contracts
First at 0x038...072Ba (memory)
globalVal
MyNewContract.sol
listen on network
Search with transaction hash or address

```
    event logString(string);
    event loguint(uint);
    event logBytes(bytes);
    event logaddress(address);

    function globalVal() public {
        emit loguint(block.gaslimit);
        emit logaddress(msg.sender);
        emit logaddress(tx.origin);
        emit logBytes(msg.data);
    }
}
```

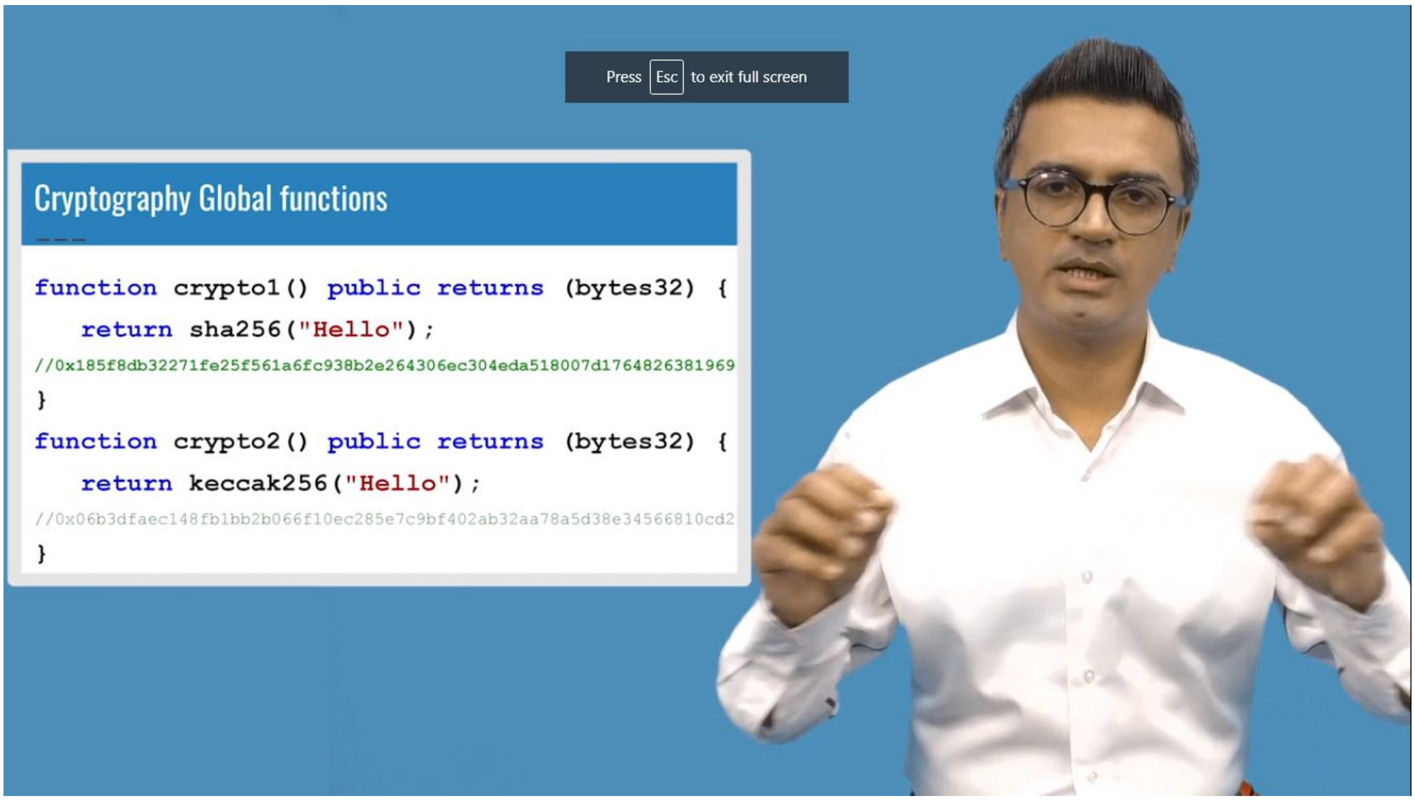
```
0: uint256: 69762765929000
```

```
[vm] from:0xca3...a733c to:First.(constructor) value:0 wei data:0x608...c0032 logs:0
hash:0xa6b...b144c

call to First.globalVal

[call] from:0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c to:First.globalVal()
data:0xfb9...e1d78
```



Cryptography Global functions

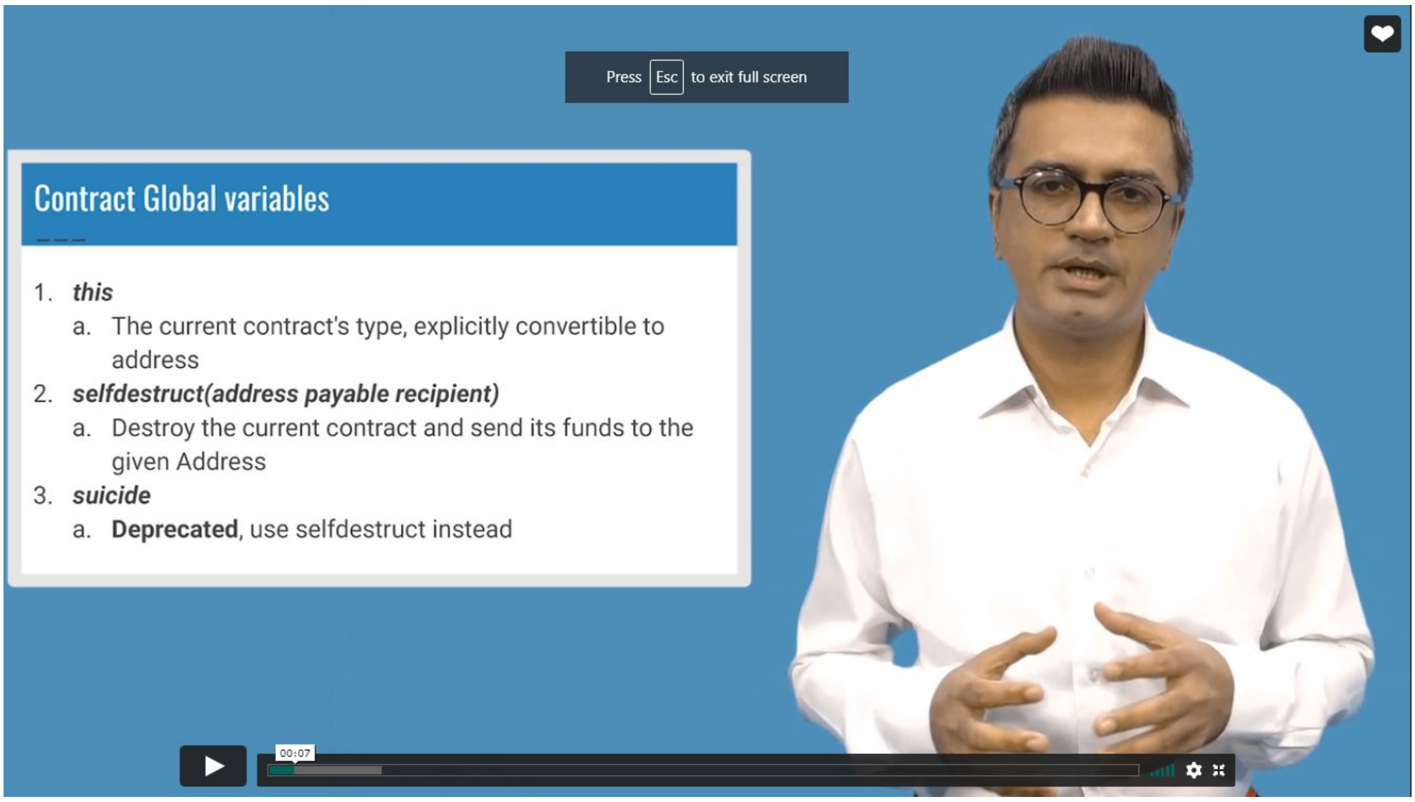
1. Solidity provides cryptographic functions for hashing values within contract functions.
2. There are two hashing function — SHA2 and SHA3.
3. sha256 function converts the input into a hash based on the sha2 algorithm
4. keccak256 function converts the input into a hash based on the sha3 algorithm
5. sha3 function has been deprecated



Cryptography Global functions

```
function crypto1() public returns (bytes32) {
   return sha256("Hello");
//0x185f8db32271fe25f561a6fc938b2e264306ec304eda518007d1764826381969
}
function crypto2() public returns (bytes32) {
   return keccak256("Hello");
//0x06b3dfaec148fb1bb2b066f10ec285e7c9bf402ab32aa78a5d38e34566810cd2
}
```

Contract Global Variables:



Contract Global variables

1. ***this***
   a. The current contract's type, explicitly convertible to address
2. ***selfdestruct(address payable recipient)***
   a. Destroy the current contract and send its funds to the given Address
3. ***suicide***
   a. **Deprecated**, use selfdestruct instead

```
contract MyBalance {

            function getEther() public payable{}

             function getValue() public returns(string memory){
                 return  'contract is working';
             }
              function removeContract() public {
                 selfdestruct(0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db);
                //address to send the ether
             }
}
```