**Solidity**

* 7. Interface
* 11. Exception handling
* 12. Hashing how to hash inside contract
* 13. Check srv or vrs
* 14. Bytes function
* 15. How to call contract from other contract (call by value)
* 2. ERC20 token (ERC721 non fungible) how it works
* 10. Inheritance
* 16. Logic for coding
* 9. Best practice to write solidity
* 8. Modifier
* Storage and memory
* constructor
* 6. External
* 5. Internal
* 4. Private
* 3. Public
* 1. Open zepline math library —> SafeMath, Math
* Sushi -swap
* Uni-swap
* Liquidity pool
* arbitrage
* Dai coin
* USDT

|  |  |
| --- | --- |
| **transaction hash** | 0xd7e82fe74148afa8bb52e6718268543e3d96113d5aa79640c2124a576ddd84c9  Copy value to clipboard |
| **from** | 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4  Copy value to clipboard |
| **to** | viewAndPure.x() 0x5A86858aA3b595FD6663c2296741eF4cd8BC4d01  Copy value to clipboard |
| **execution cost** | 23347 gas (Cost only applies when called by a contract)  Copy value to clipboard |
| **hash** | 0xd7e82fe74148afa8bb52e6718268543e3d96113d5aa79640c2124a576ddd84c9  Copy value to clipboard |
| **input** | 0x0c5...5699c  Copy value to clipboard |
| **decoded input** | {}  Copy value to clipboard |
| **decoded output** | { "0": "uint256: 2" }  Copy value to clipboard |
| **logs** | []  Copy value to clipboard |

<https://github.com/kevaundray/SmartContractChallenges/tree/master/Ethereum>

<https://www.programmersought.com/article/2720454206/>

<https://applicature.com/blog/blockchain-technology/solidity-interview>

<https://www.youtube.com/watch?v=zqHb-ipbmIo> —> overflow, underflow

<https://www.youtube.com/watch?v=XiDs_UmEDG0> —> msg.value, msg.sender, tx.origin, block.timestamp/now

<https://www.youtube.com/watch?v=TNZLonjrLYE&list=PLbbtODcOYIoE0D6fschNU4rqtGFRpk3ea&index=4> —> variable type

<https://www.youtube.com/watch?v=jPHXG82WCYA&list=PLbbtODcOYIoE0D6fschNU4rqtGFRpk3ea&index=5> —> functions

<https://www.youtube.com/watch?v=nep2-1Zzptk&list=PLbbtODcOYIoE0D6fschNU4rqtGFRpk3ea&index=8> —> view, pure, constant

<https://www.youtube.com/watch?v=xknoxALAL8c> —> pure and view

<https://www.youtube.com/watch?v=xQ987rtzWJc&list=PLbbtODcOYIoE0D6fschNU4rqtGFRpk3ea&index=10>

—> visibility of function

<https://github.com/OpenZeppelin/openzeppelin-contracts/tree/master/contracts/token> —> open zepline

<https://docs.openzeppelin.com/contracts/2.x/api/math#Math>

<https://www.youtube.com/watch?v=oTS9uxU6cAM&list=PLO5VPQH6OWdULDcret0S0EYQ7YcKzrigz&index=4> gas and gas price

<https://www.youtube.com/watch?v=BduMOagAuKs&list=PLO5VPQH6OWdULDcret0S0EYQ7YcKzrigz&index=13> events and indexed

<https://medium.com/upstate-interactive/solidity-override-vs-virtual-functions-c0a5dfb83aaf#:~:text=A%20function%20that%20allows%20an,should%20be%20marked%20as%20override%20> —> override and virtual

1. Function yourFunctionName() VisibilityModifier FunctionBehavior SpecifyTheReturnTypeOfTheFunction(variableType)

VisibilityModifier —> internal, external, public, private

Function Behavior —> view, pure, payable and constant(only used in older versions same as view)

Example: function getValue() external view returns(uint)

1. We need to change the name of variable while using in the function as it will shadow value used in contract level and won’t be able to access it inside the body of our function.

hence,

contract myContract{

uint value

// THESE ARE THE VIEW FUNCTION ONLY FOR READING

function getValue() external view returns(uint){

return value

}

//THESE ARE MODIFYING FUNCTION USED TO MODIFY OR CHANGE DATA.

function setValue(uint \_value) external { —> here we have used \_value

value = \_value

}

}

1. Every time you deploy a smart contract it creates different smart contract each associated with their address. And each smart contract won’t have any effect on other. Ie they are independent of each other if you modify one contract it won’t effect other smart contract.

Graphical user interface, application

Description automatically generated

1. View —> read only , ‘constant’ is also like view when you see constant that means you are dealing with old version of solidity
2. View, pure and payable are function behaviour
3. View —> eth\_sendTransaction (<http://man.hubwiz.com/docset/Ethereum.docset/Contents/Resources/Documents/eth_sendTransaction.html> )

Functions can be declared view if they promise not to modify the state

The following statements are considered modifying the state:

* 1. Writing to state variables.
  2. [Emitting events](https://docs.soliditylang.org/en/v0.5.3/contracts.html#events).
  3. [Creating other contracts](https://docs.soliditylang.org/en/v0.5.3/control-structures.html#creating-contracts).
  4. Using selfdestruct.
  5. Sending Ether via calls.
  6. Calling any function not marked view or pure.
  7. Using low-level calls.
  8. Using inline assembly that contains certain opcodes.

1. Solidity can detect potential state changes.
2. Pure —> eth\_call (<http://man.hubwiz.com/docset/Ethereum.docset/Contents/Resources/Documents/eth_call.html> )

Functions can be declared pure in which case they promise not to read from or modify the state.

In addition to the list of state modifying statements explained above, the following are considered reading from the state:

* 1. Reading from state variables.
  2. Accessing address(this).balance or <address>.balance.
  3. Accessing any of the members of block, tx, msg (with the exception of msg.sig and msg.data).
  4. Calling any function not marked pure.
  5. Using inline assembly that contains certain opcodes

1. Liquidity pool ( x \* y = z ) liquidity pool token  10 ,1000 1st person gets 100 liquidity token 50% 10 ,1000 2nd person gets 100 liquidity token 50%  10,1000 1st person gets 100 liquidity token 33% 10,1000 2nd person gets 100 liquidity token 33% 10,1000 3rd person gets 100 liquidity token 33%  10,1000 1st person gets 100 liquidity token 16.6 % 10,1000 2nd person gets 100 liquidity token 16.6 % 10,1000 3rd person gets 100 liquidity token 16.6 % 30,3000 4th person gets 300 liquidity token 50%  40,4000 —> after all the withdraw and fees  10,1000   x  300+x = 1/5 Number of LP token I get / total number of LP token in liquidity loop + number of token I got = Number of ETH I gave/ total number of ETH currently there + number of ETH I Gave  X/(300 + x) = 10 / 40+10 X/(300+x) = 1/5 X = 300/4  x = 75 Liquidity Token

<https://www.youtube.com/watch?v=W0Lomo8CdTM> —> ERC20 token

Calling contract from other contract:

// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

contract Base{

uint256 public value;

function setValue(uint256 \_value) public{

value = \_value;

}

function getValue() public view returns(uint256){

return value;

}

}

contract AboveBase{

Base baseContract;

function Extra() public {

baseContract = new Base();

}

function getValuefromBase() public view returns(uint256){

return baseContract.getValue();

}

function setValuefromBase(uint256 \_value) public returns(bool success){

baseContract.setValue(\_value);

return true;

}

}