Static Variable

pragma solidity ^0.4.24;

//this is the first method to initailize the value of a variable using function

contract State

{

uint public age;

function setAge() public{

age=10;

}

}

/\*

//this is the second method to initailize the value of a variable

contract State

{

uint public age =10;

}

\*/

// //this is the third method to initailize the value of a variable using constructor

// contract State

// {

// uint public age;

// constructor() public{

// age=10;

// }

// }

Local Variable

pragma solidity ^0.4.24;

//this is the first method to initailize the value of a variable using function

contract State

{

uint public age;

function setAge() public{

age=10;

}

}

/\*

//this is the second method to initailize the value of a variable

contract State

{

uint public age =10;

}

\*/

// //this is the third method to initailize the value of a variable using constructor

// contract State

// {

// uint public age;

// constructor() public{

// age=10;

// }

// }

Function

// public -- check the function outside the contract

// view -- there is no change in the variable --only just view that variable in the instance contract.

pragma solidity ^0.4.24;

contract local

{

uint age=10; // when we use public in age then there is no need to create getter function

function getter() public view returns(uint)

// if we use view then no change in the function

{

return age; // in getter function there is no change of value , then we don't have to pay for gas

}

function setter() public{

age= age+1;}

// function incre() public

// {

// age= age+1;

// }

function setter(uint newage) public

{

age= newage; // set the age of variable age using setter function assign a value

// in setter function, we change the value of variable then we pay for the gas

}

}

Static Array

pragma solidity ^0.4.24;

// fixed size array

contract Array

{

uint[4] public arr=[10,20,30,40]; //declare an array

// give the out of bound error when we increase the size of an array, fixed the no of elements

function setter(uint index, uint value) public{

arr[index] = value; // change the value of an array

}

function length() public view returns(uint)

{

return arr.length; // check the lenth of an array

}

}

Dynamic Array

pragma solidity ^0.4.24;

// fixed size array

contract Array

{

uint[] public arr; //declare an array

// give the out of bound error when we increase the size of an array, fixed the no of elements

function pushElement(uint item) public

{

arr.push(item);

// we have to push an element into the array

}

function length() public view returns(uint)

{

return arr.length; // check the lenth of an array

}

}

Boolean

pragma solidity ^0.4.24;

// contract Array{

// bool public value ;

// }

contract Array{

bool public value = true;

function check(uint a) public returns(bool)

{

if(a>100)

{

value =true;

return value;

}

else

{

value = false;

return value;

}

}

}

Mapping

pragma solidity ^0.4.24;

// mapping, concept of keys and values

//mapping(key=> value) where key is the enrolled id of student and value tells the description of the student

//like name, age, class

// key is the roll no 0 --> name is the priya, 5-> diya, 100-> shruti

// difference between the mapping and array is that we can sequential pass the value into aaray

// if we want to access3 values in mapping then we can easily do that randomly, size and complexity reduce

//in array, if we store the 3 values then we take 100 size of array and wastage of memory

contract demo

{

mapping(uint => string) public roll\_no;

function setter(uint keys, string memory value) public

{

roll\_no[keys]=value;

}

}

Loop

pragma solidity ^0.4.24;

// contract Array{

// bool public value ;

// }

contract Array{

uint[4] public arr;

uint public count;

function loop() public{

do

{

arr[count] = count;

count++;

}while(count<arr.length);

}

// function loop() public{

// for (uint i= count;i<arr.length;i++)

// {

// arr[count] = count;

// count++;

// }

// function loop() public{

// while(count<arr.length)

// {

// arr[count] = count;

// count++;

// }

// }

}