

Practical No: 07

Aim: Simple Experiments using Solidity Program Constructs (if-then, while etc...).

Theory:

❖ **Solidity**

- Solidity is an object-oriented, high-level language for implementing smart contracts. Smart contracts are programs which govern the behaviour of accounts within the Ethereum state.
- Solidity is a curly-bracket language designed to target the Ethereum Virtual Machine (EVM).
- It is influenced by C++, Python and JavaScript. You can find more details about which languages Solidity has been inspired by in the language influences section.
- Solidity is statically typed, supports inheritance, libraries and complex user-defined types among other features.
- With Solidity you can create contracts for uses such as voting, crowdfunding, blind auctions, and multi-signature wallets.
- When deploying contracts, you should use the latest released version of Solidity. Apart from exceptional cases, only the latest version receives security fixes. Furthermore, breaking changes as well as new features are introduced regularly. We currently use a 0.y.z version number to indicate this fast pace of change.

❖ **Compilation**

- A smart contract is a stand-alone script usually written in Solidity and compiled into binary or JSON and deployed to a specific address on the blockchain.
- The solidity compiler turns code into EVM bytecode, which can then be sent to the Ethereum network as a deployment transaction.
- Such deployments have more substantial transaction fees than smart contract interactions and must be paid by the owner of the contract.

❖ **Deployment Parameters**

- JavaScript VM: All the transactions will be executed in a sand box block chain in the Browser.
- Injected Provider: Remix will connect to an injectedweb3 provider. Mist and Metamask are example of providers that inject web3, thus they can be used with this option.
- Web3Provider: Remix will connect to a remote node. You will need to provide the URL address to the selected provider: geth, parity or any Ethereum client.

❖ **Additional features**

- Transferring funds from an account to the contract
- Saving the address of the contract creator
- Limiting the users' access to functions
- Withdrawing funds from the contract to an account

1. Write a program in solidity to find area and circumference of circle, rectangle and square.

Code:

```
pragma solidity ^0.5.0;
contract Shapes {
function Square(uint256 s) public pure returns (uint256, uint256) {
return (s * s, s + s);
}
function Circle(uint256 r) public pure returns (uint256, uint256) {
return ( r * r * 314/100, 2 * r * 314/100);
}
function Rectangle(uint256 l, uint256 b)public pure returns (uint256,
uint256){
return (l * b, 2 * (l + b));
}
}
```

Output:

SHAPES AT 0X9D8...A5692 (MEMORY)

Balance: 0 ETH

Circle

8

0: uint256: 200
1: uint256: 50

Rectangle

8,8

0: uint256: 64
1: uint256: 32

Square

8

0: uint256: 64
1: uint256: 16

2. Write a program in solidity to input the basic salary of an employee and calculate Gross salary according to given conditions

Basic Salary ≤ 10000 : HRA = 20%, DA = 80%

Basic Salary is between 10001 to 20000 : HRA = 25%, DA = 90%

Basic Salary ≥ 20001 : HRA = 30%, DA = 95%

Code:


```
pragma solidity ^0.5.0;

contract Salary {
    uint256 basic; // State variable
    uint256 gross;

    function BasicSalary(uint256 x) public {
        uint256 da;
        uint256 hra;
        basic = x; // Using State variable
        if (basic <= 10000) {
            da = (basic * 80) / 100;
            hra = (basic * 20) / 100;
        } else if (basic <= 20000) {
            da = (basic * 90) / 100;
            hra = (basic * 25) / 100;
        } else {
            da = (basic * 95) / 100;
            hra = (basic * 30) / 100;
        }
        gross = basic + hra + da;
    }

    function GrossSalary() public view returns (uint256) {
        return gross;
    }
}
```

Output:

▼ SALARY AT 0X5FD...9D88D (MEMORY)  

Balance: 0 ETH

BasicSalary

900000

▼

GrossSalary

0: uint256: 2025000

- 3. Write a program in solidity to create a structure student with Roll no, Name, Class, Department, Course enrolled as variables.**
- i) Add information of 5 students.**
 - ii) Search for a student using Roll no**
 - iii) Display all information**

Code:

```
pragma solidity ^0.5.0;
pragma experimental ABIEncoderV2;

contract Student {
    struct Student {
        uint256 rn;
        string name;
        string class;
        string department;
        string course;
    }

    Student[] student;
    uint256 count;

    constructor() public {
        count = 0;
    }

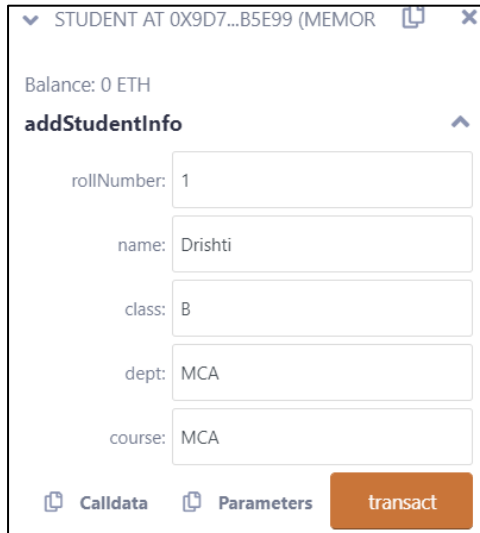
    function addStudentInfo(
        uint256 rollNumber,
        string memory name,
        string memory class,
        string memory dept,
        string memory course
    ) public {
        student.push(Student(rollNumber, name, class, dept, course));
    }

    function getStudent(uint256 rollNumber)
        public
        view
        returns (uint256, string memory)
    {
        uint256 i = 0;
        for (i = 0; i < student.length; i++) {
            if (student[i].rn == rollNumber) {
                return (student[i].rn, student[i].name);
            }
        }
        return (student[0].rn, student[0].name);
    }
}
```

```
}  
  
function displayAllInfo() public view returns (Student[] memory) {  
    return student;  
}  
}
```

Output:

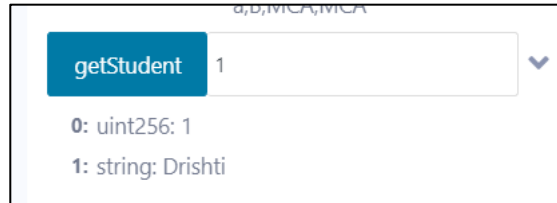
Insert Student Data:



Display Inserted Data:



Get Student name:



4. Create a structure Consumer with Name, Address, Consumer ID, Units and Amount as members. Write a program in solidity to calculate the total electricity bill according to the given condition: For first 50 units Rs. 0.50/unit
For next 100 units Rs. 0.75/unit
For next 100 units Rs. 1.20/unit
For unit above 250 Rs. 1.50/unit
An additional surcharge of 20% is added to the bill. Display the information of 5 such consumers along with their units consumed and amount.

Code:

```
pragma solidity ^0.8.0;  
pragma experimental ABIEncoderV2;  
  
contract BillCal {  
    struct Consumer {  
        uint256 units;  
        string name;  
        string addr;  
        uint256 consumerID;  
        uint256 amount;  
    }  
}
```

```
Consumer[] consumer;

function addConsumerInfo(
    uint256 units,
    string memory name,
    string memory addr,
    uint256 consumerID
) public {
    consumer.push(Consumer(units, name, addr, consumerID, 0));
}

function getBillAmount() public {
    uint256 i = 0;
    uint256 amount = 0;
    uint256 surcharge = 0;
    uint256 units = 0;
    for (i = 0; i < consumer.length; i++) {
        units = consumer[i].units;
        if (units <= 50) {
            amount = ((units * 1) / 2);
        } else if (units <= 150) {
            amount = 25 + (((units - 50) * 3) / 4);
        } else if (units <= 250) {
            amount = 100 + (((units - 150) * 6) / 5);
        } else {
            amount = 220 + (((units - 250) * 3) / 2);
        }
        surcharge = (amount * 20) / 100;
        amount = amount + surcharge;
        consumer[i].amount = amount;
    }
}

function displayConsumerInfo(uint256 consumerID)
    public
    view
    returns (
        uint256,
        string memory,
        string memory,
        uint256,
        uint256
    )
{
    uint256 i = 0;
    for (i = 0; i < consumer.length; i++) {
        if (consumer[i].consumerID == consumerID) {
            return (
```

```
        consumer[i].units,  
        consumer[i].name,  
        consumer[i].addr,  
        consumer[i].consumerID,  
        consumer[i].amount  
    );  
    }  
}  
  
return (  
    consumer[0].units,  
    consumer[0].name,  
    consumer[0].addr,  
    consumer[0].consumerID,  
    consumer[0].amount  
);  
}  
  
function displayAllInfo() public view returns (Consumer[] memory) {  
    return consumer;  
}  
}
```

Output:

▼ BILLCAL AT 0XD91...39138 (MEMORY) [copy] [x]

Balance: 0 ETH

addConsumerInfo [up arrow]

units:

name:

addr:

consumerID:

[copy] Calldata [copy] Parameters

0: tuple(uint256,string,string,uint256,uint256): 194,Drishti,Kandivali,1,182

displayConsumerInfo [up arrow]

consumerID:

[copy] Calldata [copy] Parameters

0: uint256: 194
1: string: Drishti
2: string: Kandivali
3: uint256: 1
4: uint256: 182

Conclusion: We have successfully learnt and implemented experiments using Solidity Program Constructs