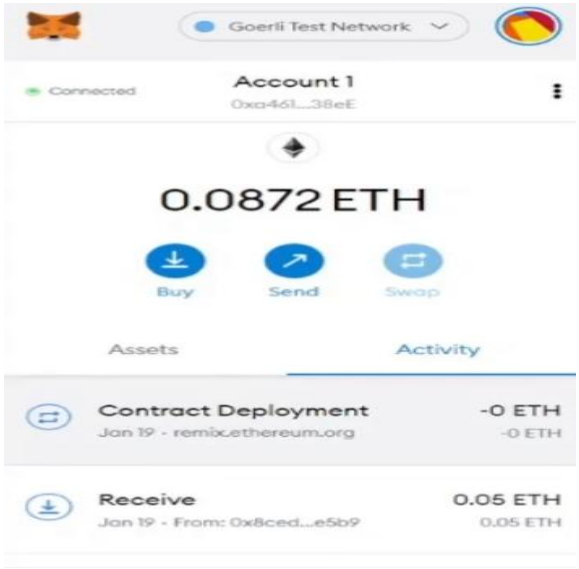
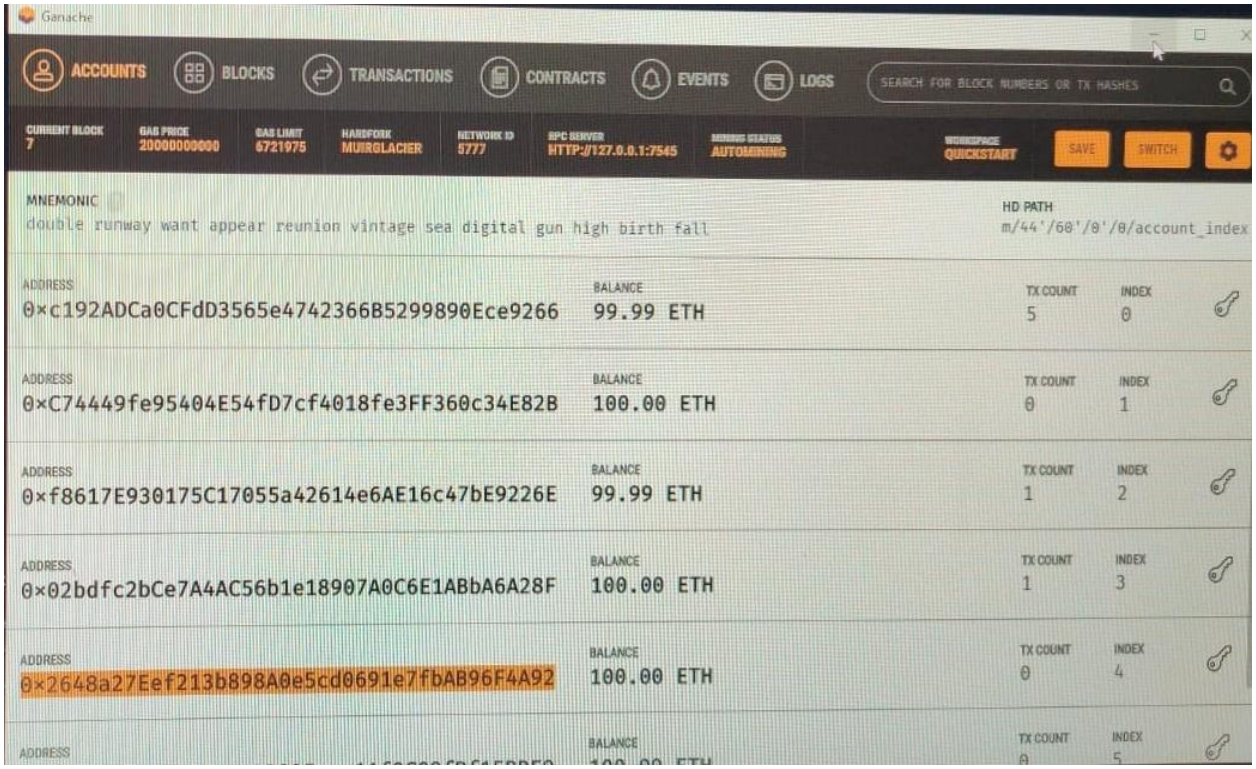
Source Code of Assignment

1. Create multiple accounts in Metamask and perform the balance transfer between the various accounts.





2. Write a solidity program to set variables and get variables.

Home testbool.sol mycontract.sol

```
1  pragma solidity^0.6.0;
2  contract myContract {
3      uint myVar;
4      function seVar(uint newVar) public {
5          myVar = newVar;
6      }
7   function getVar() public view returns (uint) {
8       return myVar;
9   }
10 }
```

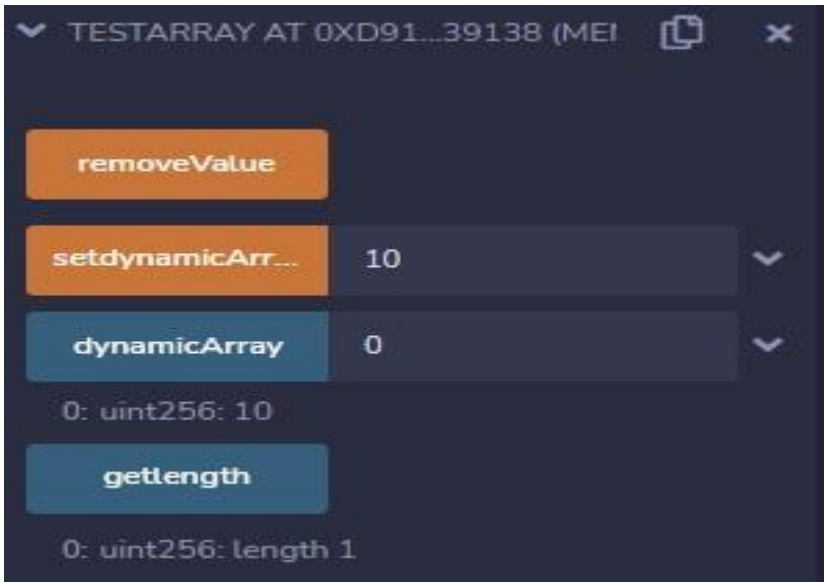
OUTPUT:

3. Write a solidity program to perform push and pop operations on dynamic array.

Home data.sol pop.sol

```
1  pragma solidity ^0.6.0;
2
3  contract testArray{
4      uint[] public dynamicArray;
5
6      function setdynamicArray(uint value) public {
7        dynamicArray.push(value);
8      }
9  function removeValue() public{
10         dynamicArray.pop();
11     }
12     function getlength() public view returns(uint length){
13         return dynamicArray.length;
14     }
15 }
```

OUTPUT:



4. Write a solidity program to set address with a mapping variable

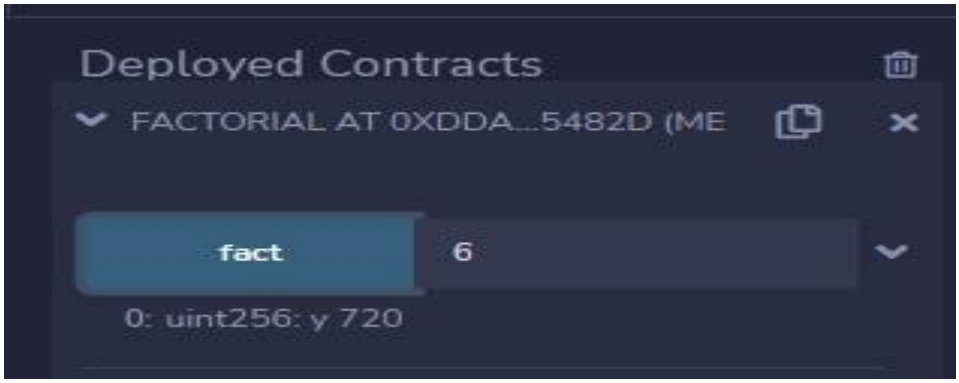
```
pragma solidity ^0.6.0;

contract testMapping{

    mapping (uint => bool) public myMapping;

    mapping (address => bool) public myAddress;

     function setValue(uint index) public{

        myMapping[index] = true;

    }

    function setAddress(address add) public {

        myAddress[msg.sender]=true;

    }

}
```

OUTPUT:

5. Write a solidity program to get the factorial of a number

```
pragma solidity ^0.6.0;

contract Factorial {

 function fact(uint x) public view returns (uint y) {

   if (x == 0) {

     return 1;

   }

   else {

    return x= x*fact(x-1);

   }

 }

}
```

OUTPUT:



6. Write a solidity program to store information of a student(Name, Roll.No, Institute, Age) using structure

```
pragma solidity 0.6.0;

contract studentdetails {
    Student[] public student;

    uint256 public studentCount;

    struct Student {
        string _firstName;
        string _lastName;
        uint256 id;
        uint256 age;
        string _InstituteName;

    }
    function addPerson(string memory _firstName, string memory _lastName, uint256 id, uint256 age, string memory _InstituteName) public {
        student.push(Student( _firstName, _lastName, id, age, _InstituteName));
    }

}
```

OUTPUT:

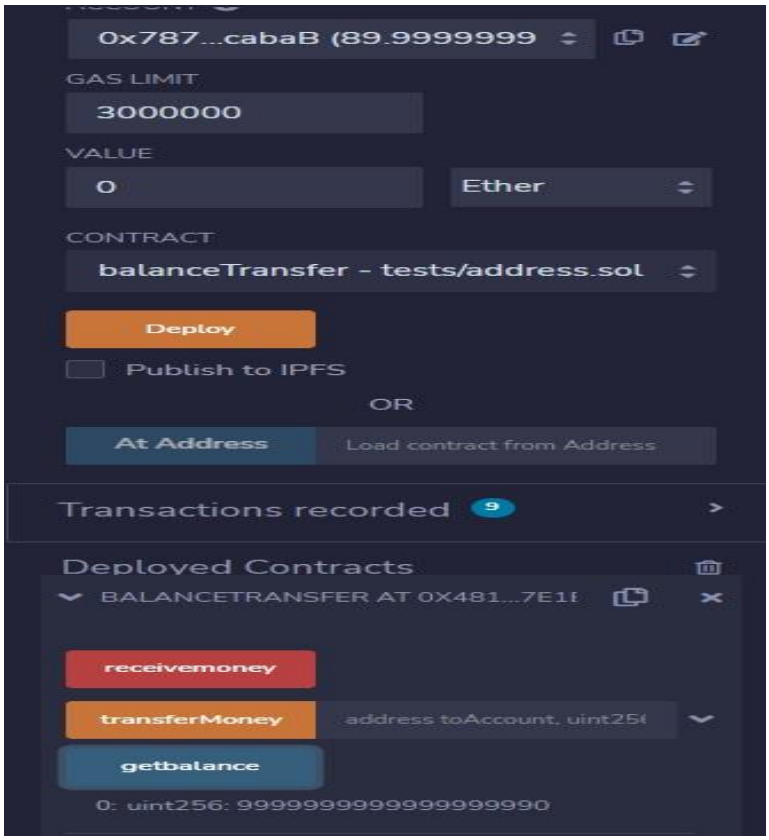
7. Write a smart contract using a solidity program to perform balance transfer from contract to other accounts.

```
&runs=200&evmVersion=null&version=soljson-v0.6.0+commit.26b70077.js
```

Home | error.sol | address.sol

```
1  pragma solidity ^0.6.0;
2
3  contract balanceTransfer{
4  function receivemoney() public payable {
5  //totalbalance += msg.value;
6  }
7  function getbalance() public view returns(uint){
8  return address(this).balance;
9  }
10 function transferMoney(address payable toAccount, uint amount) public{
11 toAccount.transfer(amount);
12 }
13 }
```

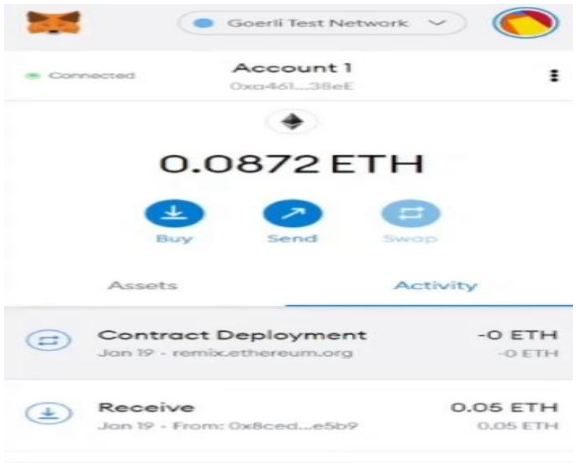
Output:



8. Write a smart contract using a solidity program to perform balance transfer with mapping and make sure only the owner can transfer the balance from contract to other contract.

```
pragma solidity ^0.6.0;

contract BalanceTransfer{
    address owner;

    mapping(address => uint) public totalBalance;

    constructor() public {
        owner = msg.sender;
    }

    function recieveBalance() public payable {
        require(msg.sender == owner, "You are not owner");
        totalBalance[msg.sender] += msg.value;
    }

    function getBalance() public view returns(uint){
        return address(this).balance;
    }

    function transferBalance(address payable toAccount, uint amount) public {
        require(msg.sender == owner, "You are not owner");
        require(totalBalance[msg.sender] >= amount, "Insufficient Balance");
        toAccount.transfer(amount);
        totalBalance[msg.sender] -= amount;
    }

}
```

Output:



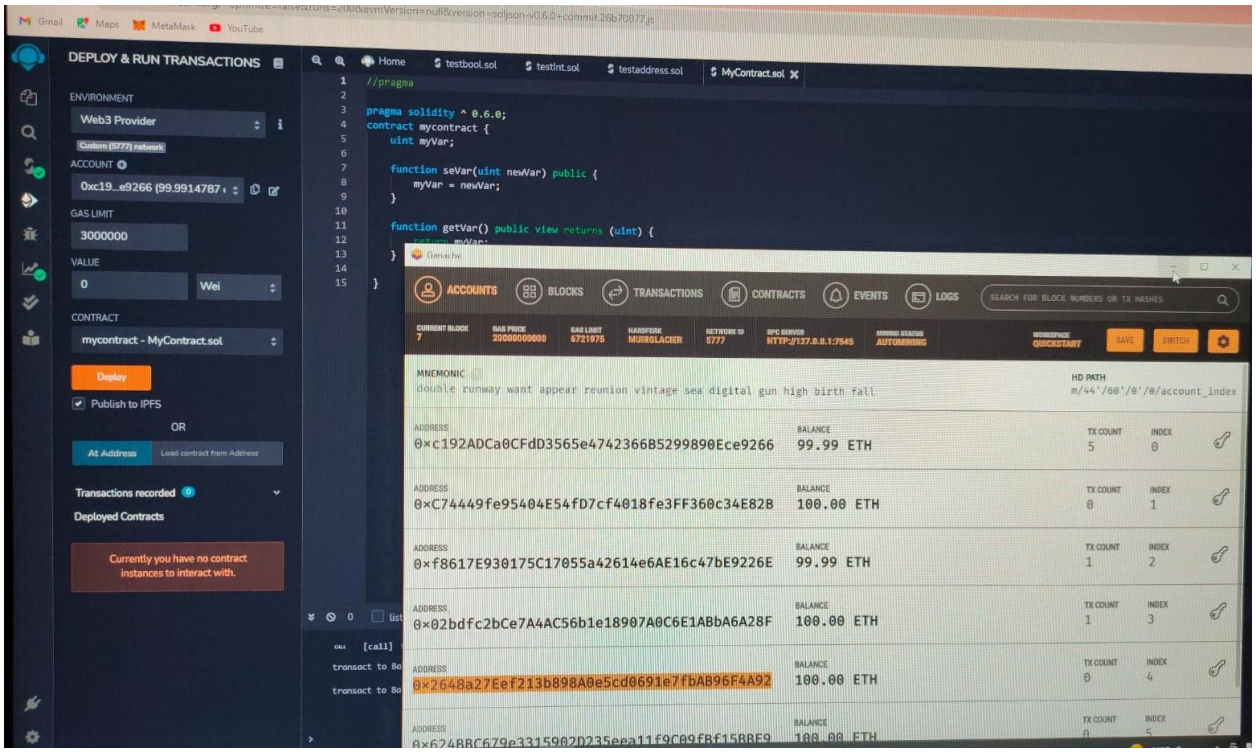
9. Write a solidity program to perform the exception handling and describe the details with screenshots.

```
pragma solidity ^0.6.0;

contract BalanceTransfer{
    address owner;

    mapping(address => uint) public totalBalance;

    constructor() public {
        owner = msg.sender;
    }

    function recieveBalance() public payable {
        require(msg.sender == owner, "You are not owner");
        totalBalance[msg.sender] += msg.value;
    }

    function getBalance() public view returns(uint){
        return address(this).balance;
    }

    function transferBalance(address payable toAccount, uint amount) public {
        require(msg.sender == owner, "You are not owner");
        require(totalBalance[msg.sender] >= amount, "Insufficient Balance");
        toAccount.transfer(amount);
        totalBalance[msg.sender] -= amount;
    }
}
```

Output:

10. Connect the following tools with the remix environment and perform balance transfer between the accounts with smart contract and share the screen shots.

a) Ganache:



b) Metamask: