1. **Create a blockchain, a genesis block and execute it.**

import datetime

import hashlib

print("Dilip Deepak Jaiswar, 05")

class MyBlock:

def \_\_init\_\_(self, previous\_hash, block\_data, timestamp):

self.previous\_hash = previous\_hash

self.block\_data = block\_data

self.timestamp = timestamp

self.hash = self.myhash()

def genesis\_block():

return MyBlock("0", "Hello World", datetime.datetime.now());

def myhash(self):

msghash = (str(self.previous\_hash) + str(self.block\_data) + str(self.timestamp))

innerblockhash = hashlib.sha256(msghash.encode()).hexdigest().encode()

Blockhash = hashlib.sha256(innerblockhash).hexdigest()

return Blockhash

b1 = MyBlock.genesis\_block(); print(b1.previous\_hash); print(b1.block\_data); print(b1.timestamp)

print(b1.hash)

num\_block = int(input("Enter the number of blocks "));

blockchain = [MyBlock.genesis\_block()]

#print("Hash is:", blockchain[0].hash)

for i in range(1, num\_block+1):

blockchain.append(MyBlock(blockchain[i-1].hash,"Good Morning", datetime.datetime.now()))

print("Hash is:", blockchain[i-1].hash)

1. **Implement and Demonstrate the Use of Solidity Programming**

**2(A)** Your First Solidity Smart Contract (Counter Program)

// SPDX-License-Identifier: GPL-3.0

pragma solidity ^0.6.0;

contract DilipCounter

{

uint count;

constructor() public

{

count = 0;

}

function getCount() public view returns(uint)

{

return count;

}

function incrementCount() public

{

count = count +1;

}

}

**2(B)** To create and explore types of variables with varying data types in solidity programming.

// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

contract DilipCalculator

{

int public num1;

int public num2;

function getnumber (int getnum1, int getnum2) public

{

num1=getnum1;

num2=getnum2;

}

function getsum() public view returns(int)

{

return num1+num2;

}

function getsub() public view returns(int)

{

return num1-num2;

}

function getmul() public view returns(int)

{

return num1\*num2;

}

function getdiv() public view returns(int)

{

return num1/num2;

}

}

**2 ( c )** Operators in solidity (Increment, Decrement)

// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

contract DilipIncrementDecrement

{

uint8 public num1;

function decrement() public

{

num1--;

}

function increment() public

{

num1++;

}

}

1. Loops in solidity

3( A) For loop in Solidity

// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

contract DilipForLoop

{

uint[] public numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

function isEvenNumber(uint number) public pure returns(bool)

{

if(number % 2 == 0)

{

return true;

}

else

{

return false;

}

}

function countEvenNumbers() public view returns (uint)

{

uint count = 0;

for(uint i =0; i < numbers.length; i++)

{

if(isEvenNumber(numbers[i]))

{

count ++;

}

}

return count;

}

}

**3(b)** While loop in Solidity

// SPDX-License-Identifier: GPL-3.0

pragma solidity ^0.6.0;

 contract DilipWhileloop

{

uint[] public numbers = [1,2,3,4,5,6,7,8,9,19];

     function countEventNumbers() public view returns (uint)

    {         uint count = 0;         uint n = 0;         while(n < numbers.length)

        {             if(isEvenNumber(numbers[n])){                 count++;

            }             n = n+1;

        }         return count;

    }

     function isEvenNumber(uint \_number) public pure returns(bool){

         if(\_number %2 == 0){

             return true;

        }         return false;

    }

}

1. Solidity Arrays and structure

**4(a)** Solidity Arrays

// SPDX-License-Identifier: GPL-3.0

pragma solidity ^0.6.0;

contract DilipArray {

 uint[] public uintArray = [22,10,1,15];

 string[] public stringArray = ['apple','watermelon','papaya', 'kiwi', 'blue berry'];

uint[][] public array2D = [ [10,20,30], [90,80,70] ];

 string[] public values;

 function addValue(string memory \_value) public {

 values.push(\_value);

 }

 function valueCount() public view returns(uint) {

 return values.length;

 }

 }

**4(b)** Structure in Solidity

// SPDX-License-Identifier: GPL-3.0

pragma solidity ^0.6.12;

 contract DilipStructure

{     struct Book

{         string title;         string author;         uint book\_id;

    }

    Book book;     function setBook() public{         book = Book("Learn Java", "TP",1);         book = Book("Learn C#","CP",2);

    }      function getBookId() public view returns(uint)

    {         return book.book\_id;

    }

}

1. Operators in Solidity

5(a) Comparison Operators

// SPDX-License-Identifier: GPL-3.0

pragma solidity ^0.6.12;

contract DilipComparison {

 // Declaring variables

    uint public a = 30;

    uint public b = 40;

    bool public equal = a == b;

    bool public notequal = a != b;

    bool public greaterthan = a > b;

    bool public lessthan = a < b;

    bool public greaterequal = a >= b;

    bool public lessequal = a <= b;

}

5(b) Logical Operators

// SPDX-License-Identifier: GPL-3.0

pragma solidity ^0.6.12;

contract DiliplogicalOperator{

     // Defining function to demonstrate

     // Logical operator

     function Logic(bool a, bool b) public pure returns (bool, bool, bool)

     {

       // Logical AND operator

       bool and = a&&b;

       // Logical OR operator

       bool or = a||b;

       // Logical NOT operator

       bool not = !a;

       return (and, or, not);

 }

}

5(c) Assignment Operators

// SPDX-License-Identifier: GPL-3.0

pragma solidity ^0.6.12;

contract DilipAssignment {

        // Declaring variables

        uint16 public assignment = 20;

        uint public assignment\_add = 50;

        uint public assign\_sub = 50;

        uint public assign\_mul = 10;

        uint public assign\_div = 50;

        uint public assign\_mod = 32;

        uint public sub;

        // Defining function to

        // demonstrate Assignment Operator

        function getResult() public{

           assignment\_add += 10;

           sub =  assign\_sub-20;

           assign\_mul \*= 10;

           assign\_div /= 10;

           assign\_mod %= 20;

           return ;

        }

}

5(d) Ternary Operators

// SPDX-License-Identifier: GPL-3.0

pragma solidity ^0.6.12;

contract DilipTernary {

 // Defining function to demonstrate conditional operator

 function conditional\_sub(uint a, uint b) public pure returns(uint) {

 uint result = (a > b? a-b : b-a);

 return result;

}

}

1. Smart contract for MLDC and Sathaye in solidity

// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

contract DilipSmartcontract {

    struct student

    {

        string name;

        string clas;

        uint256 roll;

    }

    student[21] st;

    function setstruc() public

    {

        uint256 i=0;

        while(i<=19)

        {

            st[i]=student("Dilip","MscIT",i);

            i++;

        }

        st[20]=student("Superman","Bsc",20);

    }

 //   function getnum() public (uint256)

            uint256 number;

    /\*\*

     \* @dev Store value in variable

     \* @param num value to store

     \*/

    function store(uint256 num) public {

        number = num;

      }

                uint256 first=301;

                uint256 sec=302;

                uint256 third=303;

                uint256 fourth=304;

      function check() public view returns(uint256)

      {

          if (number>0&&number <=5)

          {

              return first;

          }

          if (number>5&&number <=10)

          {

              return sec;

          }

          if (number>10&&number <=15)

          {

              return third;

          }

          if (number>15&&number <=20)

          {

              return fourth;

          }

          else

          {

              return 0;

          }

      }

      function display()public view returns(string memory)

      {

          return st[20].name;

      }

    /\*\*

     \* @dev Return value

     \* @return value of 'number'

     \*/

}

1. Mathematical Function and Function overloading in Solidity

7(a) Mathematical Function

// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

 contract DilipMathFunction

 {

     function callAddMod() public pure returns(uint)

    {

          return addmod(4, 5, 6);

    }

    function callMulMod() public pure returns(uint)

    {

        return mulmod(4, 6, 10);

    }

}

7(b) Function Overloading

// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

 contract DilipFunctionOverload

 {

     function getSum(uint a, uint b) public pure returns(uint)

     {

         return a+b;

     }

 function getSum(uint a, uint b, uint c) public pure returns(uint)

    {

        return a+b+c;

    }

    function callSumWithTwoArguments() public pure returns(uint)

    {

        return getSum(1,2);

    }

    function callSumWithThreeArguments() public pure returns(uint)

    {

        return getSum(3,4,5);

    }

}

1. Working with Account Address and Account Balance

// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

contract DilipAccountaddress {

    address public owner;

    constructor()

    {

        owner=msg.sender;

    }

    function get\_bal() public view returns(uint256)

    {

        return owner.balance;

    }

    function get\_add() public view returns(address)

    {

        return owner;

    }

}