1. ceaser Cipher:

def ceaser\_cipher(text, shift, texttype):

 cipher = ''

 for char in text:

   if char == ' ':

     cipher = cipher + char

   elif char.isupper():

    if texttype=='encrypt':

      cipher = cipher + chr((ord(char) + shift - 65) % 26 + 65)

    else:

      cipher = cipher + chr((ord(char) - shift - 65) % 26 + 65)

   else:

      if texttype=='encrypt':

        cipher = cipher + chr((ord(char) + shift - 97) % 26 + 97)

      else:

        cipher = cipher + chr((ord(char) - shift - 97) % 26 + 97)

 return cipher

plainText= input("Please enter text : ")

key= int(input("Enter key : "))

encrypted= ceaser\_cipher(plainText, key, 'encrypt')

decrypted= ceaser\_cipher(encrypted, key, 'decrypt')

print("After encryption : ", encrypted)

print("After decryption : ", decrypted)

1. **RSA**

import math

def gcd(m, n):

  if n==0:

    return abs(m)

  else:

    return gcd(n,m%n)

p = int(input("Enter a prime number(p) : "))

q = int(input("Enter a prime number(q) : "))

n = p\*q

e = int(input("Enter public key: "))

phi = (p-1)\*(q-1)

while (e < phi):

  if(gcd(e, phi) == 1):

    break

  else:

    e = e+1

d = 2 #Calculating private key : d

while d < phi:

  if ((d\*e) % phi)==1:

    break

  d += 1

pt = float(input("Enter plain text : "))

print("Plain Text = ", pt)

# Encryption c = (pt ^ e) % n

c = pow(pt, e)

c = math.fmod(c, n)

print("Encrypted data = ", c)

# Decryption m = (c ^ d) % n

m = pow(c, d)

m = math.fmod(m, n)

print("Original Message Sent = ", m)

1. SHA

import hashlib

message="Even the darkest nights will end and the sun will rise."

result = hashlib.sha256(message.encode())

print("Message : ", message)

print("Hash Value : ", result)

print("Hexadecimal equivalent of SHA256 is: ",result.hexdigest())

print("Digest Size : ", result.digest\_size)

print("Block Size : ", result.block\_size)

1. Assert:

// Solidity program to demonstrate assert statement

pragma solidity ^0.5.0;

// Creating a contract

contract assertStatement {

    // Defining a state variable

    bool result;

    // Defining a function to check condition

    function checkOverflow(uint \_num1, uint \_num2) public {

        uint sum = \_num1 + \_num2;

        assert(sum<=255);

        result = true;

    }

    // Defining a function to print result of assert statement

    function getResult() public view returns(string memory){

        if(result == true){

            return "No Overflow";

        }

        else{

            return "Overflow exist";

        }

    }

}

1. Write a Solidity to demonstrate use of “structures” for Employee

pragma solidity ^0.5.0;

pragma experimental ABIEncoderV2;

contract q3 {

    struct Employee {

        uint256 id;

        string name;

        string salary;

        string department;

    }

    Employee[] emp;

    function addEmployee(

        uint256 id,

        string memory name,

        string memory salary,

        string memory department

    ) public {

        emp.push(Employee(id, name, salary, department));

    }

    function getEmployeeByid(uint256 id)

        public

        view

        returns (uint256, string memory)

    {

        uint256 i = 0;

        for (i = 0; i < emp.length; i++) {

            if (emp[i].id == id) {

                return (emp[i].id, emp[i].name);

            }

        }

        return (0, "Not Available");

    }

    function getAllEmployee() public view returns (Employee[] memory) {

        return emp;

    }

}

1. Write a Solidity program to demonstrate view function and pure function.

pragma solidity ^0.5.0;

contract Test {

   function getResultView() public view returns(uint product, uint sum){

       uint num1 = 10;

       uint num2 = 16;

      product = num1 \* num2;

      sum = num1 + num2;

   }

   function getResultPure() public pure returns(uint product, uint sum){

      uint num1 = 2;

      uint num2 = 4;

      product = num1 \* num2;

      sum = num1 + num2;

   }

}

1. Mapping

// Solidity program to

// count number of

// values in a mapping

pragma solidity ^0.4.18;

contract mapping\_example {

    // Defining structure

    struct student {

        // Declaring different

        // structure elements

        string name;

        string subject;

        uint8 marks;

    }

    // Creating mapping

    mapping (address => student) result;

    address[] student\_result;

    //Function adding values to the mapping

    function adding\_values() public {

        var student

        = result[0xDEE7796E89C82C36BAdd1375076f39D69FafE252];

        student.name = "John";

        student.subject = "Chemistry";

        student.marks = 88;

        student\_result.push(

        0xDEE7796E89C82C36BAdd1375076f39D69FafE252) -1;

    }

    // Function to retrieve

    // values from the mapping

    function get\_student\_result(

    ) view public returns (address[]) {

        return student\_result;

    }

    // Function to count number

    // of values in a mapping

    function count\_students(

    ) view public returns (uint) {

        return student\_result.length;

    }

}

1. Write a solidity program to demonstrate creating a fixed-size array and access array element.(no metamask needed)

pragma solidity ^0.5.0;

contract Types {

    uint[6] data1;

    function array\_example() public returns (

    int[5] memory, uint[6] memory){

        int[5] memory data

        = [int(50), -63, 77, -28, 90];

        data1

        = [uint(10), 20, 30, 40, 50, 60];

        return (data, data1);

}

}

1. Patients

contract Patient {

    string patientName;

    string patientDoctor;

    string patientMedicine;

    string patientDisease;

    function get()

        public

        view

        returns (

            string memory,

            string memory,

            string memory,

            string memory

        )

    {

        return (patientName, patientDoctor, patientMedicine, patientDisease);

    }

    function set(

        string memory name,

        string memory doc,

        string memory med,

        string memory dis

    ) public {

        patientName = name;

        patientDoctor = doc;

        patientMedicine = med;

        patientDisease = dis;

    }

}

1. **Write a Solidity program to demonstrate function calling and also demonstrate return statements for multiple values.**

pragma solidity ^0.5.0;

// Creating a contract

contract Test {

   // Defining a public view function

   function return\_example() public view returns(uint, uint, uint, string memory){

      uint num1 = 10;

      uint num2 = 16;

      uint sum = num1 + num2;

      uint prod = num1 \* num2;

      uint diff = num2 - num1;

      string memory msg = "Multiple return values";

      return (sum, prod, diff, msg);

   }

}

1. Dynamic Array

// Solidity program to demonstrate

// the above approach

pragma solidity ^0.6.8;

contract DynamicArray{

// Declaring state variable

int[] private arr;

// Function to add data

// in dynamic array

function addData(int num) public

{

arr.push(num);

}

// Function to get data of

// dynamic array

function getData() public view returns(int[] memory)

{

return arr;

}

// Function to return length

// of dynamic array

function getLength() public view returns (uint)

{

return arr.length;

}

// Function to return sum of

// elements of dynamic array

function getSum() public view returns(int)

{

uint i;

int sum = 0;

for(i = 0; i < arr.length; i++)

    sum = sum + arr[i];

return sum;

}

// Function to search an

// element in dynamic array

function search(int num) public view returns(bool)

{

uint i;

for(i = 0; i < arr.length; i++)

{

    if(arr[i] == num)

    {

    return true;

    }

}

if(i >= arr.length)

    return false;

}

}