AMITY UNIVERSITY MAHARASHTRA

AMITY INSTITUTE OF INFORMATION TECHNOLOGY

BLOCKCHAIN

LAB FILE

BCA

2018-2022



Prathmesh Patil

BCA SEM5

A71004819033



AMITY UNIVERSITY MAHARASHTRA

Amity Institute of Information Technology

This is to certify that Mr. Prathmesh Patil is a Bonfire student of Amity Institute of Information Technology, at Amity University Maharashtra and he has done the project work titled “ BLOCKCHAIN LAB ” at Amity University Mumbai as prescribed by AIIT, AUM in partial fulfilment of the requirement of BCA Program for the academic year 2021-22.

Teacher’s signature :-

**INDEX:**

|  |  |  |
| --- | --- | --- |
| Index No | Experiments | Sign |
| 1 | Blockchain Code |  |
| 2 | Hello world in flask |  |
| 3 | Addition in solidity |  |
| 4 | String data type in solidity |  |
| 5 | Constructor in solidity |  |
| 6 | Factorial of 5 in solidity using loops :-  a. for  b. while  c. do while |  |
| 7 | Single Dimensional Array in solidity to calculate total of 5 number |  |
| 8 | Dynamic array in solidity |  |
| 9 | Using if and concept of pure /view functions in Solidity |  |
| 10 | Function overloading in solidity |  |
| 11 | Function overriding :  q- write a contract Person having 3 state variables :- name, address, age and 5 function set() , getname(), getage() and getaddress(). Then create another contract student which inherit the properties of contract Person . the contract student should have 1 state variable marks and set() and getmarks() function. |  |
| 12 | Write a contract which prints the area of a square and rectangle. |  |
| 13 | Access parent class function from child class function using super keyword |  |
| 14 | Create a contract called Employee having 2 state variable basic\_salary and total\_salary and 2 function set() and get() for basic\_salary. Create Another contract clerk inherit the properties of contract employee . clerk contract have 2 state variable hra and da and set(), get() -[get method to get total salary] and cal()- [cal to calculate total salary] where hra = 10% and da = 50%. Create Another contract officier inherit the properties of contract employee . officier contract have 3 state variable hra, da and sa and set(), get() -[get method to get total salary] and cal()- [cal to calculate total salary] where hra = 10% , sa=5% and da = 50%. |  |
| 15 | Create a contract called Employee having 2 state variable basic\_salary and total\_salary and 2 function constructor() and get() for basic\_salary. Create Another contract clerk inherit the properties of contract employee . clerk contract have 2 state variable hra and da and constructor (), get() -[get method to get total salary] and cal()- [cal to calculate total salary] where hra = 10% and da = 50%. Create Another contract officier inherit the properties of contract employee . officier contract have 3 state variable hra, da and sa and constructor (), get() -[get method to get total salary] and cal()- [cal to calculate total salary] where hra = 10% , sa=5% and da = 50%. |  |
| 16 | Create a structure having name, rollno and marks. . And set value using set function and display name using get function. |  |
| 17 | Create a structure having name, rollno and marks. And set value using set function and display all value of struct using get function . [pragma experimental ABIEncoderV2;] |  |
| 18 | Enum |  |
| 19 | Array of structure |  |
| 20 | Hash and modifier function |  |

1. Blockchain Code

Code:-

import string

import binascii

import Cryptodome

import Cryptodome.Hash

import datetime

import collections

import Cryptodome.Random

from hashlib import sha256

from Cryptodome.Hash import SHA

from Cryptodome.PublicKey import RSA

from Cryptodome.Signature import PKCS1\_v1\_5

class Client:

def \_\_init\_\_(clie):

random = Cryptodome.Random.new().read

clie.\_private\_key = RSA.generate(1024, random)

clie.\_public\_key = clie.\_private\_key.publickey()

clie.\_signer = PKCS1\_v1\_5.new(clie.\_private\_key)

@property

def identity(clie):

return binascii.hexlify(clie.\_public\_key.exportKey(format='DER')).decode('ascii')

class Transaction:

def \_\_init\_\_(trans, sender, recipient, value):

transactions = []

trans.sender = sender

trans.recipient = recipient

trans.value = value

trans.time = datetime.datetime.now()

trans.transactions = []

def to\_dict(self):

if self.sender == "Genesis":

identity = "Genesis"

else:

identity = self.sender.identity

return collections.OrderedDict({

'sender':identity,

'recipient':self.recipient.identity,

'value':self.value,

'time' : self.time

})

def sign\_transaction(s\_trans):

private\_key = s\_trans.sender.\_private\_key

signer = PKCS1\_v1\_5.new(private\_key)

h=SHA.new(str(s\_trans.to\_dict()).encode('utf8'))

s=binascii.hexlify(signer.sign(h)).decode('ascii')

s\_trans.transactions.append(s\_trans.to\_dict())

return s

def display\_transaction(transaction):

dict=transaction.to\_dict()

print("sender: " + str(dict['sender']))

print('-----')

print("recipient: " + str(dict['recipient']))

print('-----')

print("value: " + str(dict['value']))

print('-----')

print("time: " + str(dict['time']))

print('-----')

Prathmesh=Client()

Groot=Client()

t0=Transaction(Prathmesh,Groot,100)

t0.sign\_transaction()

t0.display\_transaction()

class Block:

valid\_transactions = []

def \_\_init\_\_(self):

#index is also required which is not shown in the code

self.valid\_transactions = []

self.previous\_block\_hash = ""

self.Nonce =0

self.hash\_digest=""

def proof\_of\_work(blk):

transactions=""

for x in range(len(blk.valid\_transactions)):

print(blk.valid\_transactions[x])

difficulty=2

#for t in blk.valid\_transactions:

#transactions+=str(t.sender.identity)+str(t.recipient)+str(t.value)+str(t.time)

message = "Hello"

for i in range(1000):

h=str(hash(message))+str(i)

#h=str(hash(transactions+blk.previous\_block\_hash+str(blk.Nonce)) + str(i)

hash\_digest=sha256(h.encode('ascii')).hexdigest()

print (hash\_digest)

print("\*\*\*\*\*\*\*\*\*\*\*\*###################")

print (i)

if hash\_digest.startswith('0' \* difficulty):

print ("after " + str(i) + " iterations found nonce: ")

break

return hash\_digest

Prathmesh = Client()

Groot = Client()

t0 = Transaction (

Prathmesh,

Groot,

500.0

)

Prathmesh = Client()

Groot = Client()

Kp = Client()

Legend= Client()

t1 = Transaction (

Prathmesh,

Legend,

100.0

)

signature = t0.sign\_transaction()

signature = t1.sign\_transaction()

block0 = Block()

#block0.previous\_block\_hash = None

#Nonce = None

block0.valid\_transactions.append(t0)

block0.valid\_transactions.append(t1)

block0.proof\_of\_work()

#print(d)

#bchain = Blockchain()

#bchain.add\_blocks(block0)

#bchain.show\_blockchain()

class Blockchain:

blk\_chain = []

last\_block\_hash = ""

def add\_blocks(new\_block, blk):

new\_block.blk\_chain.append (blk)

digest = hash (blk)

new\_block.last\_block\_hash = digest

def show\_blockchain(self):

print ("Number of blocks in the chain: " + str(len (self.blk\_chain)+1))

for x in range (len(self.blk\_chain)):

block\_temp = self.blk\_chain[x]

print ("block # " + str(x))

for transaction in block\_temp.valid\_transactions:

transaction.display\_transaction()

print ('--------------')

print ('=====================================')

ar = Client()

am = Client()

t0 = Transaction (

ar,

ar,

500.0

)

arun = Client()

amit = Client()

shashank = Client()

pawan= Client()

t1 = Transaction (

arun,

shashank,

100.0

)

signature = t0.sign\_transaction()

signature = t1.sign\_transaction()

block0 = Block()

#block0.previous\_block\_hash = None

#Nonce = None

block0.valid\_transactions.append(t0)

block0.valid\_transactions.append(t1)

block0.proof\_of\_work()

#print(d)

bchain = Blockchain()

bchain.add\_blocks(block0)

bchain.show\_blockchain()

Output:-

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

1. Hello world in flask

Code:-

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

def hello\_world():

return 'Hello, World!'

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True, use\_reloader=False )

Output:-

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, application, Word

Description automatically generated

1. Addition in solidity

Code:-

pragma solidity >= 0.7.0 < 0.9.0;

contract Storage {

  uint256 a;

  uint256 b;

  uint sum;

  function set() public{

      a=5;

      b=10;

      sum=a+b;

  }

  function get() public view returns (uint256){

      return sum;

  }

}

Output:-

A screenshot of a computer

Description automatically generated

1. String data type in solidity

Code:-

pragma solidity >=0.7.0 <0.9.0;

contract SimpleStorage{

   string storedata;

   function set() public{

       storedata="hello";

   }

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

        return storedata;

    }

}

Output:-

A screenshot of a computer

Description automatically generated

1. Constructor in solidity

Code:-

pragma solidity >=0.7.0 <0.9.0;

contract SimpleStorage{

    uint256 a;

    uint256 b;

    uint256 res;

    constructor() public{

        a=5;

        b=10;

    }

    function add() public{

        res=a+b;

    }

    function get() public view returns(uint256){

        return res;

    }

}

Output:-

A screenshot of a computer

Description automatically generated

1. Factorial of 5 in solidity using loops :-
2. for

Code:-

pragma solidity >= 0.7.0 < 0.9.0;

contract SimpleStorage1 {

    uint sum;

    function fact\_for() public view returns (uint num) {

       uint i;

        uint j=1;

        for(i=1;i<=5;i++)

        {

            j=j\*i;

        }

        return j;

    }

}

Output:-

A screenshot of a computer

Description automatically generated

1. while

Code:-

pragma solidity >= 0.7.0 < 0.9.0;

contract SimpleStorage1 {

    uint sum;

     function fact\_while() public view returns (uint num) {

        uint i=1;

        uint j=1;

        while(i<=5)

        {

            j=j\*i;

            i++;

        }

        return j;

    }

}

Output:-

A screenshot of a computer

Description automatically generated

1. do while

Code:-

pragma solidity >= 0.7.0 < 0.9.0;

contract SimpleStorage1 {

    uint sum;

    function fact\_do\_while() public view returns (uint num) {

        uint i=1;

        uint j=1;

        do

        {

            j=j\*i;

            i++;

        }while(i<=5);

        return j;

    }

}

Output:-

A screenshot of a computer

Description automatically generated

1. Single Dimensional Array in solidity to calculate total of 5 number

Code:-

pragma solidity >=0.7.0 <0.9.0;

contract SimpleStorage{

  uint [5] arr;

  uint total;

    function set() public{

       total=0;

       uint i;

      for(i=0;i<5;i++)

      {

          arr[i]=i;

      }

    }

    function cal() public {

        uint i;

        for(i=0;i<5;i++)

        {

           total = total + arr[i];

        }

    }

    function get() public view returns(uint){

        return total;

    }

}

Output:-

A screenshot of a computer

Description automatically generated

1. Dynamic array in solidity

Code:-

pragma solidity >=0.7.0 <0.9.0;

contract DynamicArray{

int[] private arr;

function addData(int num) public

{

  arr.push(num);

}

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

{

  return arr;

}

function getLength() public view returns (uint)

{

  return arr.length;

}

function getSum() public view returns(int)

{

  uint i;

  int sum = 0;

  for(i = 0; i < arr.length; i++)

    sum = sum + arr[i];

  return sum;

}

}

Output:-

A screenshot of a computer

Description automatically generated

1. Using if and concept of pure /view functions in Solidity

Code:-

Limitation of pure function :-

pragma solidity >=0.7.0 <0.9.0;

contract SimpleStorage {

    uint256 a=5;

    uint256 b=10;

function get() public  pure returns (uint) {

        uint res;

        if(a<b)

        {res=b;}

    else

{res=a;}

        return res;

    }

}

Pure function:-

pragma solidity >=0.7.0 <0.9.0;

contract SimpleStorage {

    uint256 a=5;

    uint256 b=10;

function get() public  pure returns (uint) {

        uint256 a=5;

     uint256 b=10;

uint res;

        if(a<b)

        {res=b;}

    else

{res=a;}

        return res;

    }

}

View function:-

pragma solidity >=0.7.0 <0.9.0;

contract SimpleStorage {

    uint256 a=5;

    uint256 b=10;

  //code will now run because variables a and b are not state variables and the declared inside function get()

// function get() has to be made view type

function get() public view returns (uint) {

    uint res;

        if(a<b)

           {res=b;}

        else

          {res=a;}

        return res;

    }

}

contract SimpleStorage1 {

    SimpleStorage s=new SimpleStorage();

  //code will now run because variables a and b are not state variables and theu declared inside function get()

// function get() has to be made view type

    function get() public view returns (uint) {

        uint r;

        r=s.get();

        r++;

        return r;

    }

}

Output:-

Limitation of pure function :-

A screenshot of a computer

Description automatically generated

Pure function:-

A screenshot of a computer

Description automatically generated

View function:-

A screenshot of a computer

Description automatically generated

1. Function overloading in solidity

Code:-

pragma solidity >=0.7.0 <0.9.0;

contract SimpleStorage1 {

    uint sum;

    function add(uint x, uint y) public{

        sum = x + y;

    }

    function add(uint x, uint y, uint z) public{

        sum = x + y + z;

    }

    function get() public view returns (uint num) {

        return sum;

    }

}

Output:-

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. Function overriding

q- write a contract Person having 3 state variables :- name, address, age and 5 function set() , getname(), getage() and getaddress(). Then create another contract student which inherit the properties of contract Person . the contract student should have 1 state variable marks and set() and getmarks() function.

Code:-

pragma solidity >=0.7.0 <0.9.0;

contract Person {

    string name;

    string addr;

    uint age;

    function set(string memory n, string memory ad,uint a) public{

        name=n;

        addr=ad;

        age=a;

    }

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

        return name;

    }

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

        return addr;

    }

    function getage() public view returns (uint) {

        return age;

    }

}

contract student is Person{

    uint marks;

    function set(uint m) public {

         marks=m;

    }

    function get() public view returns(uint){

        return marks;

    }

}

Output:-

A screenshot of a computer

Description automatically generated

1. Write a contract which prints the area of a square and rectange.

Code:-

pragma solidity >=0.7.0 <0.9.0;

contract Calculate {

    uint ar;

    function area(uint a) public{

        ar = a \* a;

    }

    function area(uint w, uint l) public{

        ar = l \* w;

    }

    function getrect() public view returns (uint num) {

        return ar;

    }

}

Output:-

Area of rectangle:-

A screenshot of a computer

Description automatically generated

Area of square:-

A screenshot of a computer

Description automatically generated

1. Access parent class function from child class function using super keyword

Code:-

pragma solidity >=0.7.0 <0.9.0;

contract Person {

    uint age;

    function set(uint a) public virtual {

        age=a;

    }

    function get() public virtual view returns (uint) {

        return age;

    }

}

contract student is Person{

    uint marks;

    function cal() public{

        marks = marks + super.get();

    }

    function set(uint m) public override(Person) {

        marks=m;

        super.set(10);

    }

    function get() public override(Person) view returns(uint){

        return marks;

    }

}

Output:-

A screenshot of a computer

Description automatically generated

1. Create a contract called Employee having 2 state variable basic\_salary and total\_salary

And 2 function set() and get() for basic\_salary. Create Another contract clerk inherit the properties of contract employee . clerk contract have 2 state variable hra and da and set(), get() -[get method to get total salary] and cal()- [cal to calculate total salary] where hra = 10% and da = 50%. Create Another contract officier inherit the properties of contract employee . officier contract have 3 state variable hra, da and sa and set(), get() -[get method to get total salary] and cal()- [cal to calculate total salary] where hra = 10% , sa=5% and da = 50%.

Code:-

pragma solidity >=0.7.0 <0.9.0;

contract employee{

    uint basic\_salary;

    uint total\_salary;

    function set(uint s) public virtual{

        basic\_salary=s;

    }

    function get() public virtual view returns(uint){

        return basic\_salary;

    }

}

contract clerk is employee{

    uint hra;

    uint da;

    function set(uint salary) public override(employee){

       super.set(salary);

       hra=(10\*salary)/100;

       da=(50\*salary)/100;

    }

    function cal() public{

        total\_salary= super.get()+hra+da;

    }

    function get() public override(employee) view returns(uint){

        return total\_salary;

    }

}

contract officier is employee{

    uint hra;

    uint da;

    uint sa;

    function set(uint salary) public override(employee){

       super.set(salary);

       hra=(10\*salary)/100;

       da=(50\*salary)/100;

       sa=(5\*salary)/100;

    }

    function cal() public{

        total\_salary= super.get()+hra+da+sa;

    }

    function get() public override(employee) view returns(uint){

        return total\_salary;

    }

}

Output:-

Clerk:-

A screenshot of a computer

Description automatically generated

Officer:-

A screenshot of a computer

Description automatically generated

1. Create a contract called Employee having 2 state variable basic\_salary and total\_salary

And 2 function constructor() and get() for basic\_salary. Create Another contract clerk inherit the properties of contract employee . clerk contract have 2 state variable hra and da and constructor (), get() -[get method to get total salary] and cal()- [cal to calculate total salary] where hra = 10% and da = 50%. Create Another contract officier inherit the properties of contract employee . officier contract have 3 state variable hra, da and sa and constructor (), get() -[get method to get total salary] and cal()- [cal to calculate total salary] where hra = 10% , sa=5% and da = 50%.

Code:-

pragma solidity ^0.6.6;

contract employee{

     uint basic\_salary;

     uint total\_salary;

    constructor() public{

        basic\_salary=1000;

    }

    function get() public virtual view returns(uint){

        return basic\_salary;

    }

}

contract clerk is employee{

    uint hra;

    uint da;

    constructor(uint s) public{

        basic\_salary=s;

        hra=(10\*basic\_salary)/100;

        da=(50\*basic\_salary)/100;

    }

    function cal() public{

        total\_salary= basic\_salary+hra+da;

    }

    function get() public override(employee) view returns(uint){

        return total\_salary;

    }

}

contract officier is employee{

    uint hra;

    uint da;

    uint sa;

    constructor() public{

        hra=(10\*basic\_salary)/100;

       da=(50\*basic\_salary)/100;

       sa=(5\*basic\_salary)/100;

    }

    function cal() public{

        total\_salary= basic\_salary+hra+da+sa;

    }

    function get() public override(employee) view returns(uint){

        return total\_salary;

    }

}

Output:-

Clerk:-

Text

Description automatically generated

Officer:-

Text

Description automatically generated

1. Create a structure having name, rollno and marks. . And set value using set function and display name using get function.

Code:-

pragma solidity ^0.6.6;

contract student{

    struct stud {

        string name;

        uint rollno;

        uint marks;

    }

    stud s;

    function set() public {

        s = stud("Prathmesh",33,85);

    }

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

        return s.name;

    }

}

Output:-

A screenshot of a computer

Description automatically generated

1. Create a structure having name, rollno and marks. And set value using set function and display all value of struct using get function . [pragma experimental ABIEncoderV2;]

Code:-

pragma solidity ^0.6.6;

pragma experimental ABIEncoderV2;

contract student{

    struct stud {

        string name;

        uint rollno;

        uint marks;

    }

    stud s;

    function set() public {

        s = stud("Prathmesh",33,85);

    }

    function get() public view returns(stud memory){

        return s;

    }

}

Output:-

A screenshot of a computer

Description automatically generated

1. Enum

Code:-

pragma solidity ^0.6.6;

contract student{

    struct stud {

        string name;

        uint rollno;

        uint marks;

    }

    enum state {active, inactive, cancelled}

    stud s;

    state currentState;

    function set() public {

        s = stud("Prathmesh",33,85);

    }

  function setcurrState() public {

    currentState = state.active;

    }

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

        return s.name;

    }

     function getcurrState() public view returns(state){

        return currentState;

    }

}

Output:-

A screenshot of a computer

Description automatically generated

1. Array of structure

Code:-

pragma solidity ^0.6.6;

pragma experimental ABIEncoderV2;

contract student{

    struct stud {

        string name;

        uint rollno;

        uint marks;

    }

    enum state {active, inactive, cancelled}

    stud [] s;

    stud s1;

    state currentState;

    function set(string memory name, uint rollno, uint marks) public {

        s.push(stud(name,rollno,marks));

    }

  function setcurrState() public {

    currentState = state.active;

    }

   //function change() public {

       //s1=s[0];

   //}

    function get() public view returns(stud[] memory){

        return s;

    }

     function getcurrState() public view returns(state){

        return currentState;

    }

}

Output:-

A screenshot of a computer

Description automatically generated

1. Hash and modifier function

Code:-

pragma solidity ^0.7.0;

pragma experimental ABIEncoderV2;

contract student{

struct stud {

string name;

uint rollno;

uint marks;

}

enum state {active, inactive, cancelled}

// by mapping uint rollno to Student structure, the rollno will behave as database key to entire students array

mapping(uint => stud) public students;

// faculty is the owner of the contract, only he can add students

//for making faculty as owner faculty address should be initialized to the owner of the contract as shown in the constructor

address faculty;

constructor() public {

faculty = msg.sender;

}

stud [] s;

stud s1;

 state currentState;

// A modifier function as discussed above can be used to ensure that only faculty can add students. It is created as follows

modifier Faculty() {

require(msg.sender == faculty);

\_;

}

// add declaration will also change to accommodate this

// by adding modifier name in the function declaration it is ensured that only faculty can call this function

function set(string memory name, uint rollno, uint marks) public Faculty{

s.push(stud(name,rollno,marks));

}

function setcurrState() public {

currentState = state.active;

}

 function get() public view returns(stud memory){

//s1=s[0];

return s[0];

}

function getcurrState() public view returns(state){

return currentState;

}

}

Output:-

A screenshot of a computer

Description automatically generated