//DAA PRAC 1

import java.util.\*;

public class fibonacci{

public static void main(String[] args) {

Scanner scan = new Scanner(System.in);

System.out.println("Enter the Number:");

int a=scan.nextInt();

System.out.println("Fibonacci Series using iterative method till"+" "+ a +":");

fibsimple(a);

System.out.println("\n");

System.out.println("Fibonacci Series using Recursive method till"+" "+ a +":");

System.out.print(0+" "+1);

fibrec(a,0,1,0);

System.out.println("\n");

System.out.println("Fibonacci Series using Recursive new method till"+" "+ a +":");

for(int i=0;i<a;i++){

System.out.print(" "+fibrecn(i));

}

}

private static void fibsimple(int a){

int b=0;

int c=1;

int d;

System.out.print(b+" "+c);

while(a-2>0){

d=b+c;

System.out.print(" "+d);

b=c;

c=d;

a--;

}

}

private static void fibrec(int a,int b,int c,int r){

if(a-2>0){

r=b+c;

b=c;

c=r;

System.out.print(" "+r);

fibrec(a-1, b, c, r);

}

}

private static int fibrecn(int a){

if(a<=1){

return a;

}else{

return fibrecn(a-1)+fibrecn(a-2);

}

}

}

-----------------------------------------------------------------------------------------------

//DAA PRAC 2 Huffman Coding

import java.util.\*;

public class huffman {

HashMap<Character,String> encoder;

HashMap<String,Character> decoder;

private class Node implements Comparable<Node>{

Character data;

int cost;

Node left;

Node right;

public Node(Character data,int cost){

this.data=data;

this.cost=cost;

this.left=null;

this.right=null;

}

@Override

public int compareTo(Node other){

return this.cost-other.cost;

}

}

public huffman(String input) throws Exception{

HashMap<Character,Integer> fmap = new HashMap<>();

for(char ch:input.toCharArray()){

fmap.put(ch,fmap.getOrDefault(ch,0)+1);

}

PriorityQueue<Node> pq = new PriorityQueue<>();

Set<Map.Entry<Character,Integer>> entrySet = fmap.entrySet();

for(Map.Entry<Character,Integer> entry:entrySet){

Node node = new Node(entry.getKey(), entry.getValue());

pq.offer(node);

}

while(pq.size()!=1){

Node first = pq.remove();

Node second = pq.remove();

Node newNode = new Node('\0', first.cost+second.cost);

newNode.left=first;

newNode.right=second;

pq.offer(newNode);

}

Node ft = pq.remove();

this.encoder = new HashMap<>();

this.decoder = new HashMap<>();

this.EncodeDecode(ft,"");

}

private void EncodeDecode(Node node,String osf){

if(node==null) return;

if(node.left==null && node.right==null){

this.encoder.put(node.data,osf);

this.decoder.put(osf, node.data);

}

EncodeDecode(node.left, osf+"0");

EncodeDecode(node.right, osf+"1");

}

public String Encode(String source){

String ans="";

for(char ch:source.toCharArray()){

ans = ans + encoder.get(ch);

}

return ans;

}

public String decode(String codedString){

String key="";

String ans="";

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

key = key + codedString.charAt(i);

if(decoder.containsKey(key)) {

ans = ans + decoder.get(key);

key = "";

}

}

return ans;

}

public static void main(String[] args) {

try {

String str = "My Name is Virat Kohli";

huffman huff = new huffman(str);

String cs = huff.Encode(str);

System.out.println(cs);

String dc = huff.decode(cs);

System.out.println(dc);

} catch (Exception e) {

e.printStackTrace();

}

}

}

-----------------------------------------------------------------------------------------------

//DAA Prac 3 FracKnap

import java.util.\*;

public class facKnapscak {

public class Item{

int value,weight;

public Item(int x,int y){

this.value=x;

this.weight=y;

}

}

class itemCompare implements Comparator<Item>{

@Override

public int compare(Item a, Item b){

double r1=(double)(a.value)/(double)(b.value);

double r2=(double)(a.weight)/(double)(b.weight);

if(r1<r2){

return 1;

}else if(r1>r2){

return -1;

}else{

return 0;

}

}

}

public class solve{

double fracKnapsack(int W,Item arr[],int n){

Arrays.sort(arr,new itemCompare());

int curWeight=0;

double finalvalue=0.0;

for(int i=0;i<n;i++){

if(curWeight+arr[i].weight<=W){

curWeight+=arr[i].weight;

finalvalue+=arr[i].value;

}else{

int remain = W - curWeight;

finalvalue += ((double) arr[i].value / (double) arr[i].weight) \* (double) remain;

break;

}

}

return finalvalue;

}

}

public static void main(String[] args) {

Scanner scan = new Scanner(System.in);

System.out.println("Enter The Number of Items:");

int n= scan.nextInt();

System.out.println("Enter The Weight of Knapsack:");

int w= scan.nextInt();

Item[] items = new Item[n];

for(int i=0;i<n;i++){

System.out.println("Enter The Value of Item:"+(i+1)+":");

int value = scan.nextInt();

System.out.println("Enter The Weight of Item:"+(i+1)+":");

int weight = scan.nextInt();

items[i]=new facKnapscak().new Item(value,weight);

}

solve sl = new facKnapscak().new solve();

double result =sl.fracKnapsack(w, items, n);

System.out.println("Maximum value in KnapSack is:"+result);

}

}

----------------------------------------------------------------------------------------------------------------

//DAA 4 0/1 KnapSack

import java.util.\*;

public class KnapSack {

public static class Item {

int value, weight;

public Item(int value, int weight) {

this.value = value;

this.weight = weight;

}

}

public static int knapsack(int W, Item[] items, int n) {

int[][] dp = new int[n + 1][W + 1];

for (int i = 0; i <= n; i++) {

for (int w = 0; w <= W; w++) {

if (i == 0 || w == 0) {

dp[i][w] = 0;

} else if (items[i - 1].weight <= w) {

dp[i][w] = Math.max(items[i - 1].value + dp[i - 1][w - items[i - 1].weight], dp[i - 1][w]);

} else {

dp[i][w] = dp[i - 1][w];

}

}

}

return dp[n][W];

}

public static void main(String[] args) {

Scanner scan = new Scanner(System.in);

System.out.println("Enter the number of items:");

int n = scan.nextInt();

System.out.println("Enter the weight of the knapsack:");

int W = scan.nextInt();

Item[] items = new Item[n];

for (int i = 0; i < n; i++) {

System.out.println("Enter the value of item " + (i + 1) + ":");

int value = scan.nextInt();

System.out.println("Enter the weight of item " + (i + 1) + ":");

int weight = scan.nextInt();

items[i] = new Item(value, weight);

}

int result = knapsack(W, items, n);

System.out.println("Maximum value in the knapsack: " + result);

}

}

------------------------------------------------------------------------------------------------------------------------

//DAA 5 NQueens

public class NQueens {

public static void solveNQueens(int n) {

int[][] board = new int[n][n];

if (solveNQueensUtil(board, 0, n)) {

printSolution(board);

} else {

System.out.println("No solution exists.");

}

}

public static boolean isSafe(int[][] board, int row, int col, int N) {

for (int i = 0; i < col; i++) {

if (board[row][i] == 1) {

return false;

}

}

for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) {

if (board[i][j] == 1) {

return false;

}

}

for (int i = row, j = col; i < N && j >= 0; i++, j--) {

if (board[i][j] == 1) {

return false;

}

}

return true;

}

public static boolean solveNQueensUtil(int[][] board, int col, int N) {

if (col >= N) {

return true;

}

for (int i = 0; i < N; i++) {

if (isSafe(board, i, col, N)) {

board[i][col] = 1;

if (solveNQueensUtil(board, col + 1, N)) {

return true;

}

board[i][col] = 0;

}

}

return false;

}

public static void printSolution(int[][] board) {

int N = board.length;

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

System.out.print(board[i][j] + " ");

}

System.out.println();

}

}

public static void main(String[] args) {

int N = 8;

solveNQueens(N);

}

}

-----------------------------------------------------------------------------------------------------------------------------

//DAA 6 prac quick SORT

import java.util.Random;

import java.util.Scanner;

public class Quick{

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of elements in the array: ");

int size = scanner.nextInt();

int[] arr = new int[size];

int[] copyArr = new int[size];

System.out.println("Enter the elements of the array:");

for (int i = 0; i < size; i++) {

arr[i] = scanner.nextInt();

}

System.arraycopy(arr, 0, copyArr, 0, size);

long startTime, endTime;

long deterministicTime, randomizedTime;

// Deterministic Quick Sort

startTime = System.nanoTime();

quickSortDeterministic(arr, 0, size - 1);

endTime = System.nanoTime();

deterministicTime = endTime - startTime;

System.out.println("Deterministic Quick Sort Time: " + deterministicTime + " nanoseconds");

System.out.println("Sorted Array (Deterministic):");

printArray(arr);

// Reset the array

System.arraycopy(copyArr, 0, arr, 0, size);

// Randomized Quick Sort

startTime = System.nanoTime();

quickSortRandomized(arr, 0, size - 1);

endTime = System.nanoTime();

randomizedTime = endTime - startTime;

System.out.println("Randomized Quick Sort Time: " + randomizedTime + " nanoseconds");

System.out.println("Sorted Array (Randomized):");

printArray(arr);

System.out.println("Deterministic Quick Sort is " + (double) deterministicTime / randomizedTime + " times faster.");

}

public static void quickSortDeterministic(int[] arr, int low, int high) {

if (low < high) {

int pivot = partitionDeterministic(arr, low, high);

quickSortDeterministic(arr, low, pivot - 1);

quickSortDeterministic(arr, pivot + 1, high);

}

}

public static int partitionDeterministic(int[] arr, int low, int high) {

int pivot = arr[high];

int i = low - 1;

for (int j = low; j < high; j++) {

if (arr[j] < pivot) {

i++;

swap(arr, i, j);

}

}

swap(arr, i + 1, high);

return i + 1;

}

public static void quickSortRandomized(int[] arr, int low, int high) {

if (low < high) {

int pivot = partitionRandomized(arr, low, high);

quickSortRandomized(arr, low, pivot - 1);

quickSortRandomized(arr, pivot + 1, high);

}

}

public static int partitionRandomized(int[] arr, int low, int high) {

int randomIndex = new Random().nextInt(high - low + 1) + low;

swap(arr, randomIndex, high);

return partitionDeterministic(arr, low, high);

}

public static void swap(int[] arr, int i, int j) {

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

public static void printArray(int[] arr) {

for (int num : arr) {

System.out.print(num + " ");

}

System.out.println();

}

}

---------------------------------------------------------------------------------------------------

import java.util.\*;

class Job {

char id;

int deadline, profit;

Job(char id, int deadline, int profit) {

this.id = id;

this.deadline = deadline;

this.profit = profit;

}

}

public class jobsec {

public static void printJobSequence(Job[] jobs) {

int n = jobs.length;

Arrays.sort(jobs, (a, b) -> Integer.compare(b.profit, a.profit));

int maxDeadline = 0;

for (Job job : jobs) {

if (job.deadline > maxDeadline) {

maxDeadline = job.deadline;

}

}

char[] result = new char[maxDeadline];

boolean[] slot = new boolean[maxDeadline];

Arrays.fill(result, ' ');

Arrays.fill(slot, false);

for (Job job : jobs) {

for (int i = Math.min(maxDeadline, job.deadline) - 1; i >= 0; i--) {

if (!slot[i]) {

result[i] = job.id;

slot[i] = true;

break;

}

}

}

System.out.println("Job Sequence with Maximum Profit: " + new String(result));

}

public static void main(String[] args) {

Job[] jobs = new Job[5];

jobs[0] = new Job('a', 2, 100);

jobs[1] = new Job('b', 1, 19);

jobs[2] = new Job('c', 2, 27);

jobs[3] = new Job('d', 1, 25);

jobs[4] = new Job('e', 3, 15);

System.out.println("Jobs and their Profits:");

for (Job job : jobs) {

System.out.println("Job " + job.id + ": Deadline - " + job.deadline + ", Profit - " + job.profit);

}

printJobSequence(jobs);

}

}

--------------------------------------------------------------------------------------------------------------------

#ML prac 1 uber price

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings("ignore")

data = pd.read\_csv("uber.csv")

df = data.copy()

df = df.drop(['key', 'passenger\_count','pickup\_datetime'], axis=1)

df.isnull().sum()

df.corr()

df.dropna(inplace=True)

plt.boxplot(df['fare\_amount'])

q\_low = df["fare\_amount"].quantile(0.01)

q\_hi = df["fare\_amount"].quantile(0.99)

df = df[(df["fare\_amount"] < q\_hi) & (df["fare\_amount"] > q\_low)]

df.isnull().sum()

from sklearn.model\_selection import train\_test\_split

x = df.drop("fare\_amount", axis = 1)

y = df['fare\_amount']

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.2, random\_state = 1)

from sklearn.linear\_model import LinearRegression

lrmodel = LinearRegression()

lrmodel.fit(x\_train, y\_train)

predict = lrmodel.predict(x\_test)

from sklearn.metrics import mean\_squared\_error

lrmodelrmse = np.sqrt(mean\_squared\_error(predict, y\_test))

print("RMSE error for the model is ", lrmodelrmse)

from sklearn.ensemble import RandomForestRegressor

rfrmodel = RandomForestRegressor(n\_estimators = 100, random\_state = 101)

rfrmodel.fit(x\_train, y\_train)

rfrmodel\_pred = rfrmodel.predict(x\_test)

rfrmodel\_rmse = np.sqrt(mean\_squared\_error(rfrmodel\_pred, y\_test))

print("RMSE value for Random Forest is:",rfrmodel\_rmse)

-----------------------------------------------------------------------------------------------------------------

#ML prac 2 email classifications

#!/usr/bin/env python

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score

from sklearn.neighbors import KNeighborsClassifier

df = pd.read\_csv("./emails.csv")

df.isnull().sum()

X = df.iloc[:,1:3001]

X

Y = df.iloc[:,-1].values

Y

train\_x,test\_x,train\_y,test\_y = train\_test\_split(X,Y,test\_size = 0.25)

svc = SVC(C=1.0,kernel='rbf',gamma='auto')

svc.fit(train\_x,train\_y)

y\_pred2 = svc.predict(test\_x)

print("Accuracy Score for SVC : ", accuracy\_score(y\_pred2,test\_y))

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size = 0.2, random\_state=42)

knn = KNeighborsClassifier(n\_neighbors=7)

knn.fit(X\_train, y\_train)

print(knn.predict(X\_test))

print(knn.score(X\_test, y\_test))

----------------------------------------------------------------------------------------------------------------------------

#ML prac 3 churnmodeling bank

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import seaborn as sns

sns.set()

dataset = pd.read\_csv('Churn\_Modelling.csv', index\_col = 'RowNumber')

dataset.head()

X\_columns = dataset.columns.tolist()[2:12]

Y\_columns = dataset.columns.tolist()[-1:]

print(X\_columns)

print(Y\_columns)

X = dataset[X\_columns].values

Y = dataset[Y\_columns].values

from sklearn.preprocessing import LabelEncoder

X\_column\_transformer = LabelEncoder()

X[:, 1] = X\_column\_transformer.fit\_transform(X[:, 1])

X[:, 2] = X\_column\_transformer.fit\_transform(X[:, 2])

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

pipeline = Pipeline(

[

('Categorizer', ColumnTransformer(

[

("Gender Label Encoder", OneHotEncoder(categories = 'auto', drop = 'first'), [2]),

("Geography Label Encoder", OneHotEncoder(categories = 'auto', drop = 'first'), [1])

],

remainder = 'passthrough', n\_jobs = 1)),

('Normalizer', StandardScaler())

]

)

X = pipeline.fit\_transform(X)

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size = 0.2, random\_state = 0)

from keras.models import Sequential

from keras.layers import Dense, Dropout

classifier = Sequential()

classifier.add(Dense(6, activation = 'relu', input\_shape = (X\_train.shape[1], )))

classifier.add(Dropout(rate = 0.1))

classifier.add(Dense(6, activation = 'relu'))

classifier.add(Dropout(rate = 0.1))

classifier.add(Dense(1, activation = 'sigmoid'))

classifier.summary()

classifier.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])

history = classifier.fit(X\_train, y\_train, batch\_size = 32, epochs =30 , validation\_split = 0.1, verbose = 2)

y\_pred = classifier.predict(X\_test)

print(y\_pred[:5])

y\_pred = (y\_pred > 0.5).astype(int)

print(y\_pred[:5])

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

print(cm)

print(((cm[0][0] + cm[1][1])\* 100) / len(y\_test), '% of data was classified correctly')

-------------------------------------------------------------------------------------------------------------------------------------

#ML Prac 4 diabeties

import numpy as np

import pandas as pd

data = pd.read\_csv('./diabetes.csv')

data.head()

data.isnull().sum()

for column in data.columns[1:-3]:

data[column].replace(0, np.NaN, inplace = True)

data[column].fillna(round(data[column].mean(skipna=True)), inplace = True)

data.head(10)

X = data.iloc[:, :8]

Y = data.iloc[:, 8:]

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=0)

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()

knn\_fit = knn.fit(X\_train, Y\_train.values.ravel())

knn\_pred = knn\_fit.predict(X\_test)

from sklearn.metrics import confusion\_matrix, precision\_score, recall\_score, f1\_score, accuracy\_score

print("Confusion Matrix")

print(confusion\_matrix(Y\_test, knn\_pred))

print("Accuracy Score:", accuracy\_score(Y\_test, knn\_pred))

print("Reacal Score:", recall\_score(Y\_test, knn\_pred))

print("F1 Score:", f1\_score(Y\_test, knn\_pred))

print("Precision Score:",precision\_score(Y\_test, knn\_pred))

----------------------------------------------------------------------------------------------------------------------------------------

#ML prac 5 gradient

cur\_x = 2

rate = 0.01

precision = 0.000001

previous\_step\_size = 1

max\_iters = 1000

iters = 0

df = lambda x : 2 \* (x + 3)

while previous\_step\_size > precision and iters < max\_iters:

prev\_x = cur\_x

cur\_x -= rate \* df(prev\_x)

previous\_step\_size = abs(prev\_x - cur\_x)

iters += 1

print("Local Minima Occurs at :",cur\_x)

import numpy as np

import pandas as pd

import sympy as sym

import matplotlib as pyplot

from matplotlib import pyplot

def objective(x):

return (x+3)\*\*2

def derivative(x):

return 2\*(x+3)

def gradient(alpha,start,max\_iter):

x\_list=list()

x=start

x\_list.append(x)

for i in range(max\_iter):

gradi=derivative(x)

x=x-(alpha\*gradi)

x\_list.append(x)

return x\_list

x=sym.symbols('x')

expr=(x+3)\*\*2.0

grad=sym.Derivative(expr,x)

print("{}".format(grad.doit()))

grad.doit().subs(x,2)

alpha=0.1

start=2

max\_iter=30

x=sym.symbols('x')

expr=(x+3)\*\*2

x\_cor=np.linspace(-15,15,100)

pyplot.plot(x\_cor,objective(x\_cor))

pyplot.plot(2,objective(2),'ro')

x=gradient(alpha,start,max\_iter)

x\_cor=np.linspace(-5,5,100)

pyplot.plot(x\_cor,objective(x\_cor))

x\_arr=np.array(x)

pyplot.plot(x\_arr,objective(x\_arr),'.-',color='red')

pyplot.show()

---------------------------------------------------------------------------------------------------------------------------------------

#ML Prac 6

import pandas as pd

import numpy as np

df = pd.read\_csv('./sales\_data\_sample.csv', encoding='unicode\_escape')

to\_drop = ['ADDRESSLINE1', 'ADDRESSLINE2', 'STATE', 'POSTALCODE', 'PHONE']

df = df.drop(to\_drop, axis=1)

df.isnull().sum()

df['ORDERDATE'] = pd.to\_datetime(df['ORDERDATE'])

import datetime as dt

snapshot\_date = df['ORDERDATE'].max() + dt.timedelta(days = 1)

df\_RFM = df.groupby(['CUSTOMERNAME']).agg({

'ORDERDATE' : lambda x : (snapshot\_date - x.max()).days,

'ORDERNUMBER' : 'count',

'SALES' : 'sum'

})

#Rename the columns

df\_RFM.rename(columns = {

'ORDERDATE' : 'Recency',

'ORDERNUMBER' : 'Frequency',

'SALES' : 'MonetaryValue'

}, inplace=True)

df\_RFM.head()

df\_RFM['M'] = pd.qcut(df\_RFM['MonetaryValue'], q = 4, labels = range(1,5))

df\_RFM['R'] = pd.qcut(df\_RFM['Recency'], q = 4, labels = list(range(4,0,-1)))

df\_RFM['F'] = pd.qcut(df\_RFM['Frequency'], q = 4, labels = range(1,5))

df\_RFM.head()

df\_RFM['RFM\_Score'] = df\_RFM[['R', 'M', 'F']].sum(axis=1)

df\_RFM.head()

def rfm\_level(df):

if bool(df['RFM\_Score'] >= 10):

return 'High Value Customer'

elif bool(df['RFM\_Score'] < 10) and bool(df['RFM\_Score'] >= 6):

return 'Mid Value Customer'

else:

return 'Low Value Customer'

df\_RFM['RFM\_Level'] = df\_RFM.apply(rfm\_level, axis = 1)

df\_RFM.head()

data = df\_RFM[['Recency', 'Frequency', 'MonetaryValue']]

data.head()

data\_log = np.log(data)

data\_log.head()

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaler.fit(data\_log)

data\_normalized = scaler.transform(data\_log)

data\_normalized = pd.DataFrame(data\_normalized, index = data\_log.index, columns=data\_log.columns)

data\_normalized.describe().round(2)

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.cluster import KMeans

sse = {}

for k in range(1, 21):

kmeans = KMeans(n\_clusters = k, random\_state = 1)

kmeans.fit(data\_normalized)

sse[k] = kmeans.inertia\_

plt.figure(figsize=(10,6))

plt.title('The Elbow Method')

plt.xlabel('K')

plt.ylabel('SSE')

plt.style.use('ggplot')

sns.pointplot(x=list(sse.keys()), y = list(sse.values()))

plt.text(4.5, 60, "Largest Angle", bbox = dict(facecolor = 'lightgreen', alpha = 0.5))

plt.show()

kmeans = KMeans(n\_clusters=5, random\_state=1)

kmeans.fit(data\_normalized)

cluster\_labels = kmeans.labels\_

data\_rfm = data.assign(Cluster = cluster\_labels)

data\_rfm.head()

---------------------------------------------------------------------------------------------------------------------------------------

// blockchain prac 3

// SPDX-License-Identifier: MIT

pragma solidity >= 0.7.0;

// Write a smart contract on a test network, for Bank account of a customer for

// following operations: Deposit money | Withdraw Money | Show balance

contract Bank{

mapping(address => uint) public user\_account;

mapping(address => bool) public user\_exist;

function create\_account() public payable returns(string memory){

require(user\_exist[msg.sender] == false, "Account Already created!");

user\_account[msg.sender] = msg.value;

user\_exist[msg.sender] = true;

return "Account created";

}

function deposit(uint amount) public payable returns(string memory){

require(user\_exist[msg.sender] == true, "Account not created!");

require(amount > 0, "Amount should be greater than 0");

user\_account[msg.sender] += amount;

return "Amount deposisted sucessfully";

}

function withdraw(uint amount) public payable returns(string memory){

require(user\_exist[msg.sender] == true, "Account not created!");

require(amount > 0, "Amount should be greater than 0");

require(user\_account[msg.sender] >= amount, "Amount is greater than money deposisted");

user\_account[msg.sender] -= amount;

return "Amount withdrawn sucessfully";

}

function account\_balance() public view returns(uint){

return user\_account[msg.sender];

}

function account\_exists() public view returns(bool){

return user\_exist[msg.sender];

}

}

---------------------------------------------------------------------------------------------------------------------------------------

//blockchain prac 4

// SPDX-License-Identifier: MIT

pragma solidity >=0.7.0 <0.9.0;

contract StudentManagement {

struct Student {

uint stud\_id;

string name;

string department;

}

Student[] public students;

// Event to emit when a student is added - useful for front-end applications

event StudentAdded(uint stud\_id, string name, string department);

// Function to add a new student

function addStudent(uint stud\_id, string memory name, string memory department) public {

Student memory newStudent = Student(stud\_id, name, department);

students.push(newStudent);

emit StudentAdded(stud\_id, name, department);

}

// Function to retrieve student data by ID

function getStudent(uint stud\_id) public view returns (string memory, string memory) {

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

if (students[i].stud\_id == stud\_id) {

return (students[i].name, students[i].department);

}

}

revert("Student not found");

}

// The receive function is executed on a call to the contract with empty calldata

// This is the only function that will handle Ether transfers to the contract

receive() external payable {

// Any logic that should be executed when the contract receives Ether

}

}