INDEX

Name: DHARANEESHWAR S Age: _____

Class: III CSE Sec: A Subject: _____

Address: _____

Emergency Contact : _____

Sl No	Particulars	Page No
1	Python Programs	9
2	Project : Details	9
3.	Code for DFS search	10
4.	N- Queens problem	10
5.	A* Algorithm	10
6.	Water Jug problem	10
7.	Implementation of Decision Tree classification	10
8	K-Means clustering.	10
9.	Artificial Neural Network	10
10.	Introduction to Prolog	10
11.	Prolog family tree.	10
12.	Unification and Resolution	10

completed

Python Programs: Ex: No: 1

1. Reverse the array.

```
def reverse(start, end, arr):
    while start < end:
        arr[start], arr[end] = arr[end], arr[start]
        start += 1
        end -= 1
    return arr
if __name__ == "__main__":
    arr = [1, 2, 3, 4, 5, 6, 7]
    print (reverse(0, 6, arr))
```

Output:

[7, 6, 5, 4, 3, 2, 1].

2. Monotonic:

```
def monotonic(arr, n):
    inc = True
    dec = True
    for i in range(1, n):
        if arr[i] < arr[i-1]:
            inc = False.
        if arr[i] > arr[i-1]:
            dec = False.
        if inc or dec:
            print ("monotic")
        else
            print ("not monotonic")
if __name__ == "__main__":
    arr = [6, 5, 4, 2]
    n = len(arr)
    monotonic(arr, n)
```

Output:

monotonic.

3. Substring:

```
def isSubstring(S1, S2):
    M = len(S1)
    N = len(S2)
    for i in range(N-M+1):
        j=0
        while j>M and S2[i+j]==S1[j]:
            j+=1
        if j==M:
            return True.
    return False
if __name__ == "__main__":
    S1 = "hello"
    S2 = "world hello."
    print (isSubstring (S1, S2))
```

Output:

TRUE.

4. Vowels count:

```
def vowelscount (S):
    count=0
    for i in S:
        if i in 'aeiou':
            count = count +1
        return count.
if __name__ == "__main__":
    S ="geek"
    ans = vowelscount (S)
    print (ans)
```

Output:

2.

5. Largest:

```
def largest(arr, n):
    max = arr[0]
    for i in range(1,n):
        if arr[i] > max:
            max = arr[i]
    return max.
if --name--=="--main--":
    arr = [10, 20, 30]
    n = len[arr]
    ans = largest(arr, n)
    print("largest = ", arr)
```

Output:

largest = 30.

Project: Ex No: 2

Future Investment Prediction (FIP)

Problem statement:

Most Investors find it hard to make intelligent decisions due to the differently in handling financial markets: cryptocurrencies stocks and valuable metals. They lack the tools that add in easy analysis of past and present data of the market, including the capacity of predicting future trends that help in tailoring investment strategies to maximise against within a shorter period of time. Hence, the need for this project is to work out a platform that can handle the challenges by its service of data drives insights, predictive analysis and personalised investment recommendations Based on Individual goals with request to financial and market dynamics.

Problem description:

In the high speed financial markets of today it becomes difficult for investors the make the right decisions with respect to various aspects, such as cryptocurrencies, stocks and precious Metals like gold and silver. Huge volumes of data, along with the volatile nature of markets, make it really hard for an individual investor to discover profitable opportunities and optimise has investment strategy. Besides, more than ever, timely and accurate predictions along with personalised recommendations are required. The above-discussed platform aims to empower users by processing their investment prefences and financial goals to give out strategies that maximise profit in the shortest possible time thereby bridging the gap between complicated & actionable insights.

Abstract:

It propose an all new Investment platform mastering cutting edge data analytics and machine learning to analyse historial and real time data acess cryptocurrencies, stock and platform metal - gold and silver markets. this investment platform will be capable of integrating

time series analysis, predictive modeling and sentimental analysis in a manner that devices forecasts related to prospective price Movements and market trends. Users can input the Investment amount, which, coupled with Market data, the system processes to come up with relevant, personally important Investment strategies. This tries to Max. the users profit within a time constraint, by using data driven actionable insights and optimized portfolio allocation across multiple asset.

Target Audience:

Investors needs range from those of retail investors, needing basic or sophisticated tools on investing in crypto currencies, to enthusiasts looking for more data-driven insights. Precious Metals and collectibles investors zero in on market predictions and intrinsic value. the risk-averse, including Retirees, look forward to safety, which DIY investors would like the proper tools that will help them in making decision. HNWIs pursue tailored solutions. while educational institutions implement learning platforms. Technology and AI lovers discover another frontier, innovative finance solutions.

Objective:

The objective is to create a platform that provides an accurate price prediction and personalised investment recommendation system in line with user inputs, all targeted towards securing the best possible investment strategy to be adopted in order to yeild Maximum profitably within a short period. Integrate data driven insight insight with predictive Modeling in a manner that allows users to make informed decisions towards both better financial outcomes

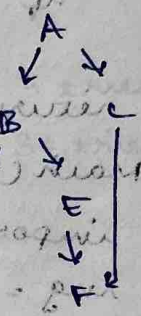
Domains:

Our fintech project focuses on leveraging innovative financial technologies and data analytics to predict market trends and optimize investments, key areas include applying machine learning, deep learning and NLP for price prediction and sentiment analysis, using quantitative finance for asset pricing and strategy development, and investing in software development to ensure robust user interfaces and data integration.

Code for DFS search : EX.NO:3

```
def dfs(graph, start, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    print(start, end=" ")
    for neighbour in graph[start]:
        if neighbour not in visited:
            dfs(graph, neighbour, visited)

graph = {
    'A' : ['B', 'C'],
    'B' : ['D', 'E'],
    'C' : ['F'],
    'D' : [],
    'E' : ['F'],
    'F' : []
}
dfs(graph, 'A')
```



Output :

A B C D E F C.

Result :

Thus, the python program for implementing DFS search Algorithm was executed successfully.

N-Queens Problem

Ex. No. 4

```
def share_diagnol(x0,y0,x1,y1):
    dx = abs(x0-x1)
    dy = abs(y0-y1)
    return dy == dx
def col_clashes(bs,c):
    for i in range(c):
        if share_diagnol(i,bs[i],c,bs[c]):
            return True
    return False
def has_clashes(the_board):
    for col in range(1,len(the_board)):
        if col_clashes(the_board,col):
            return True
    return False
def main():
    import random
    rng = random.Random()
    bd = list(range(8))
    num_found = 0
    tries = 0
    result = []
    while num_found < 10:
        rng.shuffle(bd)
        tries += 1
        if not has_clashes(bd) and bd not in
        result:
            print("Found solution {0} in {1} tri
                .format(bd,tries))
            tries = 0
            num_found += 1
            result.append(list(bd))
    print(result)
main()
```

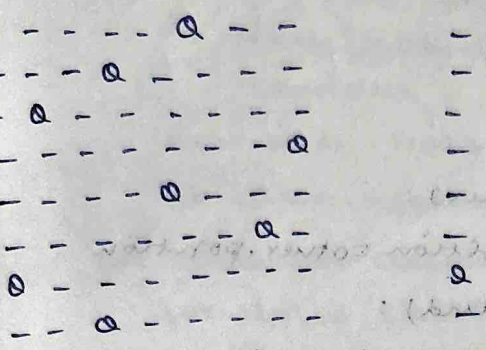
Output:

Found solution [5,3,1,7,4,6,0,2] in 688 tries

Found solution [5,2,6,1,7,4,0,3] in 421 tries

[[5,3,1,7,4,6,0,2], [5,2,6,1,7,4,0,3]]

This is the solution for 8 queens problem



for 4 queens:

found solution [1,3,0,2] in 7 tries

found solution [2,0,3,1] in 32 tries

Result:

Thus, the python program for 8-queens and 4 queens problem was executed successfully

A* Algorithm. Ex No: 5

```
import heapq
class Node:
    def __init__(self, parent=None, position=None):
        self.parent = parent
        self.position = position
        self.g = 0
        self.h = 0
        self.f = 0
    def __eq__(self, other):
        return self.position == other.position
def astar(maze, start, end):
    start_node = Node(None, start)
    end_node = Node(None, end)
    open_list = []
    closed_list = []
    open_list.append(start_node)
    while len(open_list) > 0:
        current_node = open_list[0]
        current_index = 0
        for index, item in enumerate(open_li
            if item.f < current_node.f:
                current_node = item
                current_index = index
        open_list.pop(current_index)
        closed_list.append == end_node.
        if current_node == end_node
            path = []
            current = current_node
            while current is NOT None:
                path = []
                current = current_node
                while current is not None:
                    path.append(current.
                                position)
                    current = current.parent
                return path[::-1]
            children = []
            for new_position in [(0,-1),(0,1),(-1,0),(1,0
                (-1,-1),(-1,1),(1,-1),(1,1)]:
                node_position = (current_node.
                                 position[0] +
```

```
            new_position[0], current_node.position[1]+
            new_position[1])
            if node_position[0] > (len(maze)-1) or
                node_position[1] > (len(maze[len(maze)
                -1])-1)
            or
            continue
        if maze[node_position[0]][node_position[1]] !=
            continue.
        new_node = Node(current_node, node_position)
        children.append(new_node)
    for child in children:
        for closed_child in closed_list:
            if child == closed_child:
                continue
        child.g = current_node.g+1
        child.h = ((child.position[0]-end_node.position
                                    [0])**2)
            +((child.position[1]-end_node.position[1])
                                    **2)
        for open_node in open_list:
            if child == open_node and child.g >
                                    open_node.g:
                continue
        open_list.append(child)
def main():
    maze = [[0,0,0,0,1,0,0,0,0,0]
            [0,0,0,0,1,0,0,0,0,0]
            [0,0,0,0,1,0,0,0,0,0]
            [0,0,0,0,1,0,0,0,0,0]
            [0,0,0,0,1,0,0,0,0,0]
            [0,0,0,0,0,0,0,0,0,0]
            [0,0,0,0,1,0,0,0,0,0]
            [0,0,0,0,1,0,0,0,0,0]
            [0,0,0,0,1,0,0,0,0,0]
            [0,0,0,0,0,0,0,0,0,0]]

start = (0,0)
end = (7,6)
path = astar(maze, start, end)
print(path)
if __name__ == __main__:
    main()
```

Output :

[(0,0),(1,1),(2,2),(3,3),(4,3),(5,4),(6,5),(7,8)]

Result :

Thus, the python program for A* algorithm is verified and executed successfully

Water Jug: Ex.No:6.

Program code:

```
def fill_4_gallon(x, y, x_max, y_max):
    return (x_max, y)
def fill_3_gallon(x, y, x_max, y_max):
    return (x, y_max)
def empty_4_gallon(x, y, x_max, y_max):
    return (0, y)
def empty_3_gallon(x, y, x_max, y_max):
    return (x, 0)
def pour_4_to_3(x, y, x_max, y_max):
    transfer = min(x, y_max - y)
    return (x - transfer, y + transfer)
def pour_3_to_4(x, y, x_max, y_max):
    transfer = min(y, x_max - x)
    return (x + transfer, y - transfer)
def dfs_water_jug(x_max, y_max, goal_x,
                  visited=None, start=(0,0)):
    if visited is None:
        visited = set()
    stack = [start]
    while stack:
        state = stack.pop()
        x, y = state
        if state in visited:
            continue
        visited.add(state)
        print(f"Goal reached: {state}")
        return state
    next_states = [fill_4_gallon(x, y, x_max, y_max),
                   fill_3_gallon(x, y, x_max, y_max),
                   empty_4_gallon(x, y, x_max, y_max),
                   empty_3_gallon(x, y, x_max, y_max),
                   pour_4_to_3(x, y, x_max, y_max),
                   pour_3_to_4(x, y, x_max, y_max)]
    for new_state in next_states:
        if new_state not in visited:
            stack.append(new_state)
    return None
```

x-max = 4

y-max = 3

goal-x = 2

dfs-water-jug (x-max, y-max, goal-x)

OUTPUT:

Visiting state : (0, 0)
Visiting state : (0, 3)
Visiting state : (3, 0)
Visiting state : (3, 3)
Visiting state : (4, 2)
Visiting state : (4, 0)
Visiting state : (1, 3)
Visiting state : (1, 0)
Visiting state : (0, 1)
Visiting state : (4, 1)
Visiting state : (3, 2)
Visiting state : (2, 3)

Result : Thus the python program for water jug problem is executed successfully.

Implementation of Decision Tree classification Techniques.

Ex.No.7

Aim:

To implement a decision tree classification technique for gender classification using Python.

Explanation:

→ Import tree from sklearn

→ call the function DecisionTreeClassifier() from Tree

→ Assign values for x and y

→ call the function predicting on the Basics of given random values for each given feature.

→ Display the output

Program:

```
from sklearn import tree
X = [[5.8,150],[5.2,120],[6.6,180],[5.4,125],[5.9,160],
     [5,1,110],[6.1,200],[5.3,140],[5.7,155],
     [6.2,210]]
Y = [1,0,1,0,1,0,1,0,1,1]
classifier = tree.DecisionTreeClassifier()
classifier = classifier.fit(X,Y)
sample_data = [[5.5,145]]
prediction = classifier.predict(sample_data)
gender = "Male" if prediction[0] == 1 else "Female"
print(f"The predicted gender for the input
      {sample_data} is: {gender}")
```

Output:

The predicted gender for the Input [[5.5,145]] is: Female.

Result:

Thus, the program to Implement decision tree is executed successfully.

k-means clustering

Ex. No:

Aim:

To implement a k-means clustering technique using python language.

Explanation:

→ Import k Means from sklearn.cluster

→ Assign x and y

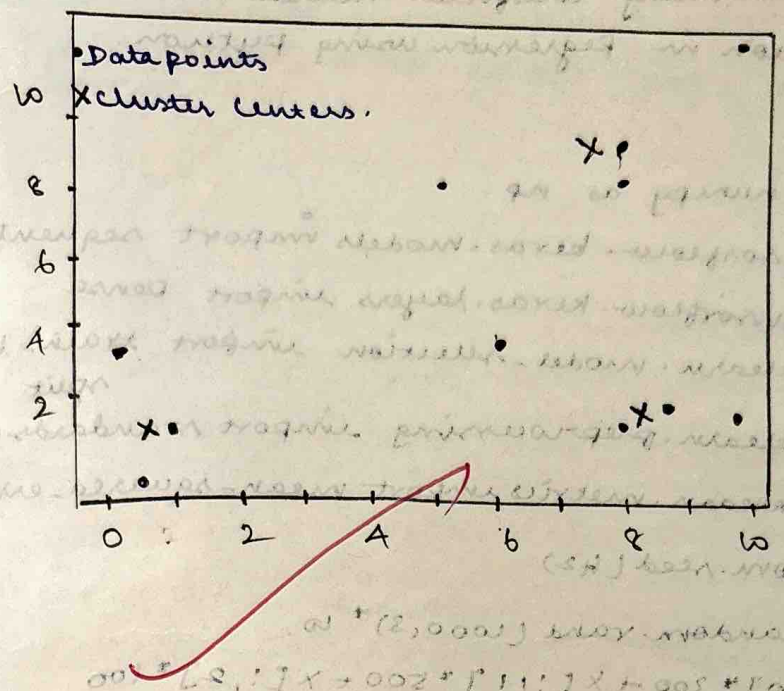
→ call the function kMeans()

→ Perform scatter operation, and display the output.

Program:

```
from sklearn.cluster import kmeans
import numpy as np
import matplotlib.pyplot as plt
x=np.array([1,2],[1.5,1.8],[5,8],[8,8],[1,0.6],
    [9,11],[8,2],[10,2],[9,3],[8,9],[0,3],[6,4]).
kmeans = kmeans(n_clusters=3)
kmeans.fit(x)
centers= kmeans.cluster_centers
labels = kmeans.labels
plt.scatter(x[:,0],x[:,1], c=labels,cmap='viridis'
marker='o', label='Data points')
plt.scatter(centers[:,0],centers[:,1],s=200,
c='red', marker='x', label='Cluster Centers')
plt.xlabel("x-axis")
plt.ylabel("y-axis")
plt.legend()
plt.show()
```

Output:



Result: Thus an algorithm using k-means was successfully executed

Artificial Neural Networks. Ex. No: 9

Aim:

To implementing Artificial Neural Networks for an Application in Regression using Python.

Program:

```
import numpy as np
from tensorflow.keras.models import sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import standardscaler
from sklearn.metrics import mean_squared_error.
np.random.seed(42)
X = np.random.rand(1000,3)*10.
Y = X[:,0]*300 + X[:,1]*500 + X[:,2]*100
    +5000 + np.random.randn(1000)*100.
X_train, X_test, Y_train, Y_test = train_test_
split (X, Y, test_size=0.2, random_state=42)
scaler = standardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
model = Sequential()
model.add (Dense(64, input_dim = X.shape[1],
    activation = 'relu'))
model.add (Dense(64, input_dim = 'relu'))
model.add (Dense(1))
model.compile (optimizer 'admin', loss = 'mse',
metrics = ['mae'])
,history = model.fit (X_train, Y_train, epochs=100,
batch_size = 32, validation_split = 0.2,
verbose = 1)
loss, mae = model, evaluate (X_test, Y_test,
verbose = 0)
print ("Mean absolute error on test data:",
    mae)
Y_pred = model.predict (X_test)
mse = mean_squared_error (Y_test, Y_pred)
print ("Mean squared error on test
        data :", mse)
```

Output:

Mean Absolute error on test data : 717.3563842773438

Mean squared error on test data : 729649.3500198255

Result: Thus a Artificial Neural Networks was successfully implemented

Introduction to Prolog. Ex: No: 1

Aim:

To learn prolog terminologies and write basic programs.

Terminologies:

1. Atomic terms:

Atomic terms are usually strings made up of lower and upper case letter, digits and the underscore with a lowercase letter.

Eg: dog
ab_c_321

2. Variables:

Variables are strings of letter, digits, and the underscore, starting with a capital letter or an underscore.

Ex: Dog
Apple_420.

3. Compound Terms:

Compound Terms are made up of a PROLOG atom and a number of arguments (PROLOG terms. i.e, atoms, Numbers, variables, or other compound terms) enclosed in parantheris and separated by commas.

Ex: is_bigger(elephant,X)
f(g(X,_),7)

4. Facts:

A fact is a predicate followed by a dot

Ex: bigger_animal(whale)
life_is_beatiful.

5. Rules:

A rule consists of a head (a predicate) and a body (a sequence of predicates separated by commas)

Ex: is_smaller(X,Y):- is_bigger(Y,X)
aunt(Aunt,Child):- sister(Aunt,Parent),
parent(Parent,Child)

source code :

KB1 :

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody)
party.
```

Query 1: ?- woman(mia).
Query 2: ?- playsAirGuitar(mia).
Query 3: ?- party
Query 4: ?- concert.

output :

```
?- woman(mia).
true
?- playsAirGuitar(mia).
false
?- party.
true
?- concert.
ERROR: Unknown procedure: concert/0 (DWIM could not
                                    correct goal)
```

KB2 :

```
happy(yolanda)
listens2music(mia)
listens2music(yolanda) :- happy(yolanda).
playsAirGuitar(mia) :- listens2music(mia)
playsAirGuitar(yolanda) :- listens2music(yolanda)
```

output :

```
?- playsAirGuitar(mia)
true
?- playsAirGuitar(yolanda).
true.
```

KB3 :

```
likes(dan, sally).
likes(sally, dan).
likes(john, brittney).
married(X,Y) :- likes(X,Y), likes(Y,X).
friends(X,Y) :- likes(X,Y); likes(Y,X).
```

Output:

```
?- likes(dan,X)
X = sally
?- married(dan,sally)
true.
?- married(john,brittney)
false.
```

KB4:

```
food(burger).
food(sandwich).
food(pizza).
lunch(sandwich).
dinner(sandwich).
dinner(pizza).
meal(X):-food(X).
```

Output:

```
?- food(Pizza)
True
?- meal(X),lunch(X)
X=sandwich
?- dinner(sandwich)
false.
```

KB5:

```
owns(jack,car(bmw)).
owns(jack,car(chevy)).
owns(olivia,car(civic)).
owns(jane,car(chevy)).
sedan(car(bmw)).
sedan(car(civic)).
truck(car(chevy)).
```

Output:

```
?-owns(john,X).
X=car(chevy)
?-owns(john,_)
true
?-owns(who,car(chevy))
who=john
?-owns(jane,X),truck(X)
X=car(chevy)
```

Result:

Thus the Basic problems to learn prolog technologies has been executed successfully.

Prolog - Family Tree. Ex. NO: 11

Aim:

To develop a family tree program using PROLOG with all possible facts, rules and queries.

Source code:

Knowledge Base:

```
/* FACTS :: */
male(peter).
male(john).
male(chris).
male(kevin).

female(betty).
female(jeny).
female(lisa)
female(helen).
parentOf(chris, peter).
parentOf(chris, betty).
parentOf(helen, peter).
parentOf(helen, betty).
parentOf(kevin, chris).
parentOf(kevin, lisa)
parentOf(jeny, john).
parentOf(jeny, helen).

/* Rules :: */
/* son, parent
* son, grandparent */
father(X,Y) :- male(Y), parentOf(X,Y)
mother(X,Y) :- female(Y), parentOf(X,Y)
grandmother(X,Y) :- male(Y), parentOf(X,Z),
                    parentOf(Z,Y)
grandmother(X,Y) :- female(Y), parentOf(X,Z),
                    parentOf(Z,Y)
brother(X,Y) :- male(Y), father(X,Z), father(Y,W), Z==W
sister(X,Y) :- female(Y), father(X,Z), father(Y,W),
               Z==W.
```

Result: Thus the prolog problems to implement and execute family tree was successfully completed

Unification and Resolution

Ex. No.

Aim:

To execute programs based on unification and resolution.

Ex: Let us see for below the prolog program - how unification and instantiation take place after querying.

FACTS: likes(john, jane)
likes(jane, john)

Query: ?- likes(john, X)
X = jane.

Here upon asking the query first prolog to search matching terms in Facts in top-down manner for likes predicate with with arguments and it can match likes(john, ...) i.e. Unification, then it looks for the value of X asked in query and it returns Answer X = jane, i.e - Instantiation - X is instantiated to jane.

Ex: 2. At the prolog query prompt, when you write below query.

?- owns(X, car(bmw)) = owns(Y, car(C))

You will get answer: X = Y, C = bmw.

Here owns(X, car(bmw)) and owns(Y, car(C)) unifies -- because i) predicate names own are same on both side - ii) No. of arguments for that predicate i.e 2 are are equal both side iii) 2nd argument with car predicate inside the brackets are same both side and even in that predicate again number of arguments are same. So, here terms unify in which X = Y, so, Y is substituted with X -- i.e written as {X | Y} and C is instantiated to bmw, -- written as {bmw | C} and this is called unification with Instantiation.

Resolution is one kind of proof technique that works this ways

i) select two clauses that contain conflicting terms
ii) combine those two clauses
iii) cancel out the conflicting terms.

For example we have following statements.

(i) If it is a pleasant day then you will do strawberry-picking

(ii) If you are doing strawberry picking you are happy

Above statements can be written in propositional logic like this:

(i) strawberry-picking ← pleasant

(ii) happy ← strawberry-picking.

And again these statements can be written in CNF like this:

(i) (strawberry-picking ∨ ¬pleasant) ∧

(ii) (happy ∨ ¬ strawberry-picking)

By resolving these two clauses and cancelling out the conflicting terms strawberry-picking and ¬strawberry-picking, we can have one new clause

(3) ¬pleasant ∨ happy

How? see the figure on right

When we write above new clause in infer or implies form, we have pleasant → happy or happy ← pleasant.

i.e. If it is a pleasant day then you are happy.

(i) (strawberry-picking ∨ ¬pleasant)

(ii) (happy ∨ ¬ strawberry-picking)

(3) ¬pleasant ∨ happy.

But sometimes from the collection of the statements we have, we want to know the answer of this question - "Is it possible to prove some other statements from what we actually know?" In order to prove this, we need to make some interfaces and other statements can be shown true using Refutation proof Method i.e proof by contradiction using Resolution. So for the asked goal we will negate the goal and will add it to the given statements to prove the contradiction.

Lets see an example to understand how Resolution and Refutation work. In below example, Part (I) represents the English meanings for the clauses, Part (II) represents the propositional logic statements Part (III) represents the conjuctive normal form (CNF) to Part (II) and part (IV) shows some other

statements we want to prove using Refutation proof Method.

Part (I): English sentences.

(1, If it is sunny and warm day you will enjoy
(2) If it is warm and pleasant day you will do strawberry picking.
(3) If it is raining then no strawberry picking
(4, If it is raining you will get wet.
(5) It is warm day.
(6) It is raining
(7) It is sunny.

Part (II): propositional statements.

(1) enjoy ← sunny ∧ warm
(2) strawberry-picking ← warm ∧ pleasant
(3) ¬strawberry-picking ← raining.
(4) wet ← raining.
(5) warm
(6) raining
(7) sunny.

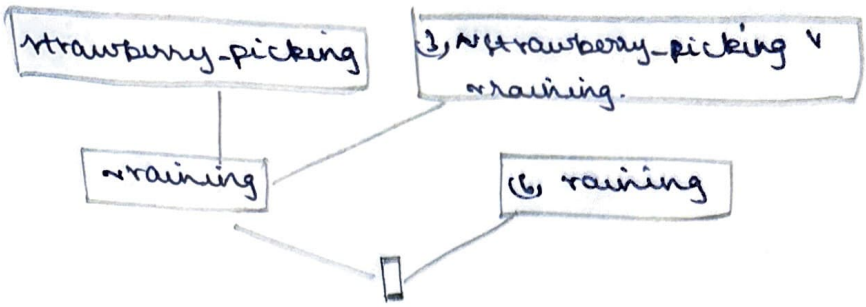
Part (III): CNF of part (II)

(1) (enjoy ∨ ¬sunny ∨ ¬warm) ∧
(2) (strawberry-picking ∨ ¬warm ∨ ¬pleasant) ∧
(3) (¬strawberry-picking ∨ ¬raining) ∧
(4) (wet ∨ ¬raining)
(5) (warm) ∧
(6) (raining) ∧
(7) (sunny).

Part (IV): Other statements we want to prove by Refutation.

(Goal 1): You are not doing strawberry picking
(Goal 2): You will enjoy
(Goal 3): Try it yourself: You will get wet.

Goal 1: You are not doing strawberry picking
Prove: ¬strawberry-picking
Assume: strawberry-picking (negate the goal, and add it to given clauses).



Goal 2: You will enjoy
Prove: enjoy
Assume: ¬enjoy (negate the goal and add it to given clauses).

Source code:

```
enjoy: - sunny, warm.
strawberry-picking: - warm, pleasant
not strawberry-picking: - raining
wet: - raining
 warm.
 raining.
 sunny.
```

Output:

```
? - not strawberry-picking.
true
? - enjoy.
 true
? - wet.
 true.
```

Result:

Thus the programs Based on unification and resolution has Been executed successfully.