INDEX

Name P.DHARSHINI

Standard Section Roll No. 220701065

School/College Subject

completed

PYTHON PROGRAMS:

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("monotonic")
        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 =" worldhello"
    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
```

Ex:02

N-Queens Problem

```
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} tries.".
                        format(bd, tries))
            tries = 0
            num_found += 1
            result.append(list(bd))
print(result)
```

Ex:01

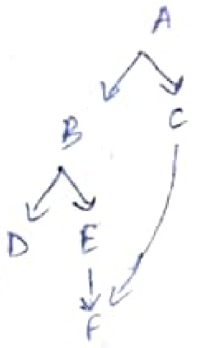
Code for DFS search :

```
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 D E F C
```



Result:

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

KB2

```
happy(yolanda).
listen2music(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, brittany)
false
```

KB4:

```
food(burger)
food(sandwich)
food(pizza)
```

~~food(~~

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

```
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 : 717.3563348713433
Mean squared Error on test data : 729649.3560198205
```

RESULT:

Thus, the program to implement ANN for an application in regression using python is executed successfully.

Ex:04

Water Jug:

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_3to_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
```

EX:06 IMPLEMENTATION OF K-MEANS CLUSTERING TECHNIQUE

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_cluster=3)
kmeans.fit(x)
centers = kmeans.cluster_centers_
labels = kmeans.labels_
plt.scatter(x[:,0], x[:,1], c=labels, cmap=[illegible]
marker='o', label='Data points')
plt.scatter(centers[:,0], centers[:,1], s=200
c='red', marker='x', label='cluster centers')
plt.title("K-Means clustering")
plt.xlabel("x-axis")
plt.ylabel("Y-axis")
plt.legend()
plt.show()
```

Ex:03

A* Algorithm

```
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_list):
            if item.f < current_node.f:
                current_node = item
                current_index = index

        open_list.pop(current_index)
        closed_list.append(current_node)

        if current_node == end_node:
            path = []
            current = current_node
            while current is not None:
                path.append(current.position)
                current = current.parent
            return path[::-1]
```

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

```
- - - - - Q - -        - - - - - Q - -
- - - Q - - - -        - - Q - - - - -
- Q - - - - - -        - - - - - - Q -
- - - - - - - Q        - Q - - - - - -
- - - - Q - - -        - - - - - - - Q
- - - - - - Q -        - - - - Q - - -
Q - - - - - - -        Q - - - - - - -
- - Q - - - - -        - - - Q - - - -
```

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

```
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[0]
       <0 or node.position[1] >(len(maze[len(maze)-1])-1)
     or
        continue
     if maze[node_position[0]][node_position[1]] != 0:
        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]
```

```
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_3to_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 : (2, 3)
Visiting state : (2, 3)
```

the python program for water jug problem

successfully.

```
[0,0 0, 0,0,  0,0, 0, 0, 0,]
[0,0,0,0,1, 0 0 9 0, 9 ]
[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,
```

Result:

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

EX:05 Implementation of Decision Tree classification Techniques

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], [5.2,130], [6.0, 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, 10, 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.

EX:07 IMPLEMENTATION OF ARTIFICIAL NEURAL NETWORKS FOR APPLICATION IN REGRESSION

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, xtest, 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 (32, activation = 'relu'))
model.add (Dense(1))
model.compile (optimizer 'adam', loss='mse',
  metrics = ['mae'])
history = model.fit (x_train, y train, epochs=15,
 batch_size = 32, validation_split=0.2, verbose =1)
```

OUTPUT

RESULT:

[illegible]

[illegible]

[illegible]

INTRODUCTION TO PROLOG

EX:08

AIM:

To learn PROLOG terminologies and write basic programs

TERMINOLOGIES:

1. Atomic Terms:

Atomic terms are usually strings made up of lower- and uppercase letters, digits and the underscore, starting with a lowercase letter.

Ex: dog
ab_c_321

2. variables:

variables are strings of letters, 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 parenthesis and seperated by commas

Ex

```
is_bigger(elephant, x)
f(g(x,_),7)
```

OUTPUT:

```
?- food(pizza)
   true
?- meal(x), lunch(x)
 x= sandwich
? - dinner(sandwich)
  false
```

KB5:

```
owns(jack, car(bmw)).
owns(john, 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(who, car(chevy)).
  who= john
?- owns(john, _).
  true
?- owns(jane, x), sedan(x).
  false
?- owns(jane, x), truck(x).
  x= car(chevy).
```

RESULT:

Thus the basic program on PROLOG terminologies are executed successfully.

4. Facts:

A fact is a predicate followed by a dot.

Ex.

bigger_animal(whale)

life is beautiful.

5. Rules

A rule consist 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
```

11 M

PROLOG FAMILY TREE

AIM:

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

CODE:

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

female(betty).
female(jenny).
female(lisa).
female(helen)

parentof (chris, peter).
parentof (chris, betty).
parent of (helen, peter).
parent of (kevin, chris).
parent of (helen, betty).
parentof (kevin, lisa).
parentof( jeny, john).
parentof (jeny, helen].

/* Rule::*/
/* son, parent
* son, grandparent
father(X,Y):-male(Y), parentof (X,Y)
mother(X,Y):- female(Y), parentof(X,Y)
```

```
granfather(X,Y):- male(Y), parentof(X,Z), parentof(Z,Y)
grandmother(X,Y):- female(Y), parentof(X,Z), parentof(Z,Y)
mother(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
```

Output:

```
? parentof(kevin, X)
  X = chris

? father(X, chris)
  X = kevin

? sister(X, chris)
  false
```

RESULT:

Thus, the PROLOG program to implement and execute family tree was successfully completed.