**Ex.No:10**

**Date:**

**IMPLEMENTATION OF LOGIC PROGRAMMING PARADIGM**

**AIM:**

To implement logic programming paradigm in python.

### 1. Write the python program to map values to a function using pyDatalog

**ALGORITHM 1:**

* Import pyDatalog and pyDatalog
* Creating new pyDatalog variables
* Creating square function
* Reverse order as desired
* Displaying the output .
* Print the output.

**PROGRAM 1:**

from pyDatalog import pyDatalog

pyDatalog.create\_terms("A, Pow2, Pow3")

def square(n):

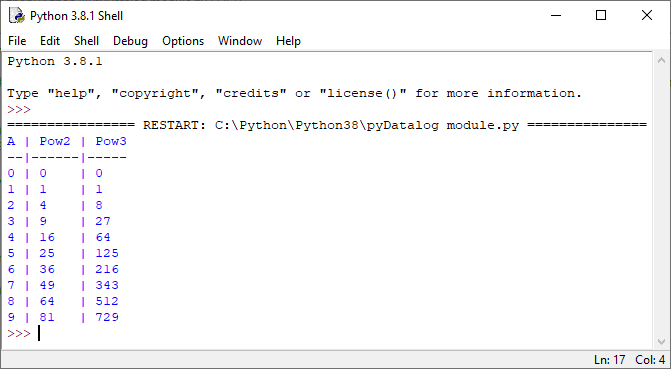
return n\*n

pyDatalog.create\_terms("square")

input\_values = range(10)[::-1]

print ( A.in\_(input\_values) & (Pow2 == square(A)) & (Pow3 == A\*\*3) )

**OUTPUT 1:**

****

2. Write the python program to solve 8 Queens problem

**ALGORITHM 2:**

* Import pyDatalog
* The queen in the first column can be in any row
* Find the queens in the first 2 columns, find the first one first, then find a second one
* Repeat for the queens
* The second queen can be in any row, provided it is compatible with the first one
* Find the third queen, first find a queen compatible with the second one, then with the first
* Rre-use the previous clause for maximum speed, thanks to memorization
* Repeat for all queens
* Keep one queen in row X1 and another in row X2 if they are separated by N columns
* Print the queens

**PROGRAM 2 :**

from pyDatalog import pyDatalog

pyDatalog.create\_terms('N,X0,X1,X2,X3,X4,X5,X6,X7')

pyDatalog.create\_terms('ok,queens,next\_queen')

# the queen in the first column can be in any row

queens(X0) <= (X0.\_in(range(8)))

# to find the queens in the first 2 columns, find the first one first, then find a second one

queens(X0,X1) <= queens(X0) & next\_queen(X0,X1)

# repeat for the following queens

queens(X0,X1,X2) <= queens(X0,X1) & next\_queen(X0,X1,X2)

queens(X0,X1,X2,X3) <= queens(X0,X1,X2) & next\_queen(X0,X1,X2,X3)

queens(X0,X1,X2,X3,X4) <= queens(X0,X1,X2,X3) & next\_queen(X0,X1,X2,X3,X4)

queens(X0,X1,X2,X3,X4,X5) <= queens(X0,X1,X2,X3,X4) & next\_queen(X0,X1,X2,X3,X4,X5)

queens(X0,X1,X2,X3,X4,X5,X6) <= queens(X0,X1,X2,X3,X4,X5) & next\_queen(X0,X1,X2,X3,X4,X5,X6)

queens(X0,X1,X2,X3,X4,X5,X6,X7) <= queens(X0,X1,X2,X3,X4,X5,X6) & next\_queen(X0,X1,X2,X3,X4,X5,X6,X7)

# the second queen can be in any row, provided it is compatible with the first one

next\_queen(X0,X1) <= queens(X1) & ok(X0,1,X1)

# to find the third queen, first find a queen compatible with the second one, then with the first# re-use the previous clause for maximum speed, thanks to memoization

next\_queen(X0,X1,X2) <= next\_queen(X1,X2) & ok(X0,2,X2)

# repeat for all queens

next\_queen(X0,X1,X2,X3) <= next\_queen(X1,X2,X3) & ok(X0,3,X3)

next\_queen(X0,X1,X2,X3,X4) <= next\_queen(X1,X2,X3,X4) & ok(X0,4,X4)

next\_queen(X0,X1,X2,X3,X4,X5) <= next\_queen(X1,X2,X3,X4,X5) & ok(X0,5,X5)

next\_queen(X0,X1,X2,X3,X4,X5,X6) <= next\_queen(X1,X2,X3,X4,X5,X6) & ok(X0,6,X6)

next\_queen(X0,X1,X2,X3,X4,X5,X6,X7) <= next\_queen(X1,X2,X3,X4,X5,X6,X7) & ok(X0,7,X7)

# it's ok to have one queen in row X1 and another in row X2 if they are separated by N columns

ok(X1, N, X2) <= (X1 != X2) & (X1 != X2+N) & (X1 != X2-N)

# give me one solution to the 8-queen puzzle

print(queens(X0,X1,X2,X3,X4,X5,X6,X7).data[0])

(4, 2, 0, 6, 1, 7, 5, 3)

**OUTPUT 2 :**

(7, 3, 0, 2, 5, 1, 6, 4)

(4, 2, 0, 6, 1, 7, 5, 3)

**RESULT:**

Thus the Python program to implement logical programming paradigm have been written and executed successfully.

**Ex.No:11**

**Date:**

**IMPLEMENTATION OF DEPENDENT TYPE PROGRAMMING PARADIGM**

**AIM:**

To implement Dependent Type Programming paradigm in python.

1. Write the python program to Program to create type aliases typing module.

**ALGORITHM 1:**

* Start
* From typing import list
* Initiate vector e
* def scale function initialize to vector :

Return [ scalar\* number of or number in vector]

* Declare a scale function
* Print output
* End the program

**PROGRAM 1:**

from typing import List

# Vector is a list of float values

Vector = List[float]

def scale(scalar: float, vector: Vector) -> Vector:

return [scalar \* num for num in vector]

a = scale(scalar=2.0, vector=[1.0, 2.0, 3.0])

print(a)

**OUTPUT 1 :**

[2.0, 4.0, 6.0]

2. Program to check every key: value pair in a dictionary and check if they match the name : email format.

**AIM:**

To write a program to check every key: value pair in a dictionary and check if they match the name : email format.

**ALGORITHM 2:**

* Start
* From typing import disk
* import rc
* create an alias called ‘Contact Diet’
* check if name and strings
* check for email
* Print the output

**PROGRAM 2:**

from typing import Dict

import re

# Create an alias called 'ContactDict'

ContactDict = Dict[str, str]

def check\_if\_valid(contacts: ContactDict) -> bool:

for name, email in contacts.items():

# Check if name and email are strings

if (not isinstance(name, str)) or (not isinstance(email, str)):

return False

# Check for email xxx@yyy.zzz

if not re.match(r"[a-zA-Z0-9\.\_\+-]+@[a-zA-Z0-9\.\_-]+\.[a-zA-Z]+$", email):

return False

return True

print(check\_if\_valid({'viji': 'viji@sample.com'}))

print(check\_if\_valid({'viji': 'viji@sample.com', 123: 'wrong@name.com'}))

**OUTPUT 2:**

True

False

**ALGORITHM 3:**

* Start
* From typing import New Type
* Create a new user Type called ‘Student ID’ that consists of an integer.
* Create a new user Type called ‘Student ID’
* define a function get\_student\_name.
* return username for ID.
* Print student details.

**PROGRAM 3:**

from typing import NewType

# Create a new user type called 'StudentID' that consists of

# an integer

StudentID = NewType('StudentID', int)

sample\_id = StudentID(100)

from typing import NewType

# Create a new user type called 'StudentID'

StudentID = NewType('StudentID', int)

def get\_student\_name(stud\_id: StudentID) -> str:

return str(input(f'Enter username for ID #{stud\_id}:\n'))

stud\_a = get\_student\_name(StudentID(100))

print(stud\_a)

stud\_b = get\_student\_name(-1)

print(stud\_b)

**OUTPUT 3**

Enter username for ID #100:

Joseph

Joseph

Enter username for ID #-1:

John

John

**RESULT:**

Thus the Dependent type programming paradigm programs has been executed successfully.

**Ex.No:12**

**Date:**

**IMPLEMENTATION OF NETWORK PROGRAMMING PARADIGM**

**AIM:**

To implement network programming paradigm in python.

1. Program to implement client server program using TCP

### ALGORITHM 1:

### TCP Server:

* Import Socket
* Create a socket object
* Reserve a port on your computer
* Next bind to the port this makes the server listen to requests coming from other computers on the network
* Put the socket into listening mode
* Establish connection with client.
* Send a thank you message to the client.
* Close the connection with the client
* Print the output.

### TCP Client:

* Import Socket
* Create a socket object
* Define the port on which you want to connect
* Connect to the server on local computer
* Receive data from the server
* Close the connection with the server
* Print the output

**PROGRAM 1:**

**TCP Server**

#!/usr/bin/python # This is server.py file

import socket # Import socket module

s = socket.socket() # Create a socket object

host = socket.gethostname() # Get local machine name

print(host)

port = 12348 # Reserve a port for your service.

s.bind((host, port)) # Bind to the port

s.listen(5) # Now wait for client connection.

print('Socket successfully created')

print('socket binded to 12348')

print('socket is listening')

while True:

c, addr = s.accept() # Establish connection with client.

print('Got connection from', addr)

c.send(b'Thank you for connecting')

c.close() # Close the connectionimport socket

**OUTPUT 1:**

Socket successfully created

socket binded to 12348

socket is listening

Got connection from ('127.0.0.1', 52617)

**TCP Client**

#!/usr/bin/python # This is client.py file

import socket # Import socket module

s = socket.socket() # Create a socket object

host = socket.gethostname() # Get local machine name

port = 12348 # Reserve a port for your service.

s.connect((host, port))

print(s.recv(1024))

s.close()

**OUTPUT 1:**

Thank you for connecting

2. Program to implement client server program using UDP

### ALGORITHM 2

### UDP Server:

* Import Socket
* Create a datagram socket
* Bind to address and ip address
* Listen for incoming datagram
* Send reply to client
* Print client message
* Print the client IP address.

### UDP Client:

* Import Socket
* Create a client socket
* Create a UDP socket at client side
* Send to server using created UDP socket
* Print the server message

**UDP Server:**

import socket

localIP = "192.168.1.100"

localPort = 20002

bufferSize = 1024

msgFromServer = "Hello UDP Client"

bytesToSend = str.encode(msgFromServer)

# Create a datagram socket

UDPServerSocket = socket.socket(family=socket.AF\_INET, type=socket.SOCK\_DGRAM)

# Bind to address and ip

UDPServerSocket.bind((localIP, localPort))

print("UDP server up and listening")

# Listen for incoming datagrams

while(True):

bytesAddressPair = UDPServerSocket.recvfrom(bufferSize)

message = bytesAddressPair[0]

address = bytesAddressPair[1]

clientMsg = "Message from Client:{}".format(message)

clientIP = "Client IP Address:{}".format(address)

print(clientMsg)

print(clientIP)

# Sending a reply to client

UDPServerSocket.sendto(bytesToSend, address)

**OUTPUT 2:**

UDP server up and listening

Message from Client:Hello UDP Server

Client IP Address:('192.168.1.100', 52397)

**UDP Client**

import socket

msgFromClient = "Hello UDP Server"

bytesToSend = str.encode(msgFromClient)

serverAddressPort = ("192.168.1.100", 20002)

bufferSize = 1024

# Create a UDP socket at client side

UDPClientSocket = socket.socket(family=socket.AF\_INET, type=socket.SOCK\_DGRAM)

# Send to server using created UDP socket

UDPClientSocket.sendto(bytesToSend, serverAddressPort)

msgFromServer = UDPClientSocket.recvfrom(bufferSize)

msg = "Message from Server {}".format(msgFromServer[0])

print(msg)

**OUTPUT 2:**

Message from Server Hello UDP Client

**RESULT:**

Thus the Python program to implement TCP client and server and UDP client and server have been written and executed successfully.