

INDEX

Name Madhasoodan Reddy

Sub. AI-Lab

Std.: V

Div. Sem

Roll No. 1BM21CS099

Telephone No.

E-mail ID.

Blood Group.

Birth Day.

Sr.No.	Title	Page No.	Sign./Remarks
3/11/23 1	Python Introduction		
10/11/23 2	Python programs		
17/11/23 3	Tic Tac Game	60 17/11/23	
24/11/23 4	8-Puzzle Game, BFS	61 24/11/23	
07			
08/11/23 5	8 - Puzzle using A* algorithm	62 3/12/23	
6	8 - puzzle using IDDFS	63 3/12/23	
22/12/23 7	Vaccum cleaner.	64 22/12/23	
29/12/23 8	Knowledge Base	65 29/12/23	
18/1/24			
19/01/24 1) Implement using FOL		10 24/1/24	
2) FOL to PNF conversion.		10 24/1/24	
3) create KB consists of FOL & prove the given query using forward reasoning.		10 24/1/24	

3/11/23

Lab-1

S-135

PAGE NO.:

DATE:

3/11/23

Python :-

Topics Learnt:-

- 1) Introduction : (ppa) initiation - ppa 99b
- 2) Functions and Getting Help.
- 3) Booleans and conditionals
- 4) Lists : (S1 => ppa) 1.119
- 5) Loops and List comprehensions
- 6) Strings and Dictionaries : (ppa) 99c 4.119
- 7) Working with External Libraries.

(("mississippi") trireq

((("n") * 999 + "m" * 3) * 1) trireq

(ppa) strings - ppa

- tuple

ppa 999 + m * 3

11

abhi

10/11/23

Lab - 2

PAGE NO.:

DATE:

Write Python Programs

Program

2) Program for age categorisation

```
def age_criteria(age):
    if (age <= 2):
        print("Infants")
    elif (age <= 12):
        print("Kids")
    elif (age <= 20):
        print("Teens")
    elif (age <= 59):
        print("Adults")
    else:
        print("Senior citizens")
```

```
age = int(input("Enter the age \n"))
age_criteria(age)
```

Output:-

Enter the age

22

Kids

Lab - 2

10/11/23

Write Python Programs

Please

2) Program for age categorisation

```
def age_criteria(age):
    if (age <= 2):
        print("Infants")
    elif (age <= 12):
        print("Kids")
    elif (age <= 20):
        print("Teens")
    elif (age <= 50):
        print("Adults")
    else:
        print("Senior citizens")
```

```
age = int(input("Enter the age \n"))
age_criteria(age)
```

Output:-

Enter the age

22

Kids

PAGE NO:

DATE:

PAGE NO:

DATE:

2) Program for multiplication tables :-

```
def multi_table(num,k):
    for i in range(k):
        ans = num * i + 1
        print(f'{num} * {i+1} = {ans} \n')
```

```
table = int(input("Enter the number \n"))
```

```
k = int(input("Enter till which number \n"))
```

```
multi_table(table, k)
```

Output:-

Enter the number Enter till which number

5

5x1=5

5x2=10

5x3=15

5x4=20

5x5=25

5x6=30

5x7=35

5x8=40

5x9=45

5x10=50

(5 times) printing 50

(i) prints 50 5 times

(i+1) prints 50 6 times

(i+2) prints 50 7 times

(i+3) prints 50 8 times

(i+4) prints 50 9 times

(i+5) prints 50 10 times

(i+6) prints 50 11 times

(i+7) prints 50 12 times

(i+8) prints 50 13 times

(i+9) prints 50 14 times

(i+10) prints 50 15 times

(i+11) prints 50 16 times

(i+12) prints 50 17 times

(i+13) prints 50 18 times

(i+14) prints 50 19 times

(i+15) prints 50 20 times

(i+16) prints 50 21 times

(i+17) prints 50 22 times

(i+18) prints 50 23 times

(i+19) prints 50 24 times

(i+20) prints 50 25 times

(i+21) prints 50 26 times

(i+22) prints 50 27 times

(i+23) prints 50 28 times

(i+24) prints 50 29 times

(i+25) prints 50 30 times

(i+26) prints 50 31 times

(i+27) prints 50 32 times

(i+28) prints 50 33 times

(i+29) prints 50 34 times

(i+30) prints 50 35 times

(i+31) prints 50 36 times

(i+32) prints 50 37 times

(i+33) prints 50 38 times

(i+34) prints 50 39 times

(i+35) prints 50 40 times

(i+36) prints 50 41 times

(i+37) prints 50 42 times

(i+38) prints 50 43 times

(i+39) prints 50 44 times

(i+40) prints 50 45 times

(i+41) prints 50 46 times

(i+42) prints 50 47 times

(i+43) prints 50 48 times

(i+44) prints 50 49 times

(i+45) prints 50 50 times

(i+46) prints 50 51 times

(i+47) prints 50 52 times

(i+48) prints 50 53 times

(i+49) prints 50 54 times

(i+50) prints 50 55 times

(i+51) prints 50 56 times

(i+52) prints 50 57 times

(i+53) prints 50 58 times

(i+54) prints 50 59 times

(i+55) prints 50 60 times

(i+56) prints 50 61 times

(i+57) prints 50 62 times

(i+58) prints 50 63 times

(i+59) prints 50 64 times

(i+60) prints 50 65 times

(i+61) prints 50 66 times

(i+62) prints 50 67 times

(i+63) prints 50 68 times

(i+64) prints 50 69 times

(i+65) prints 50 70 times

(i+66) prints 50 71 times

(i+67) prints 50 72 times

(i+68) prints 50 73 times

(i+69) prints 50 74 times

(i+70) prints 50 75 times

(i+71) prints 50 76 times

(i+72) prints 50 77 times

(i+73) prints 50 78 times

(i+74) prints 50 79 times

(i+75) prints 50 80 times

(i+76) prints 50 81 times

(i+77) prints 50 82 times

(i+78) prints 50 83 times

(i+79) prints 50 84 times

(i+80) prints 50 85 times

(i+81) prints 50 86 times

(i+82) prints 50 87 times

(i+83) prints 50 88 times

(i+84) prints 50 89 times

(i+85) prints 50 90 times

(i+86) prints 50 91 times

(i+87) prints 50 92 times

(i+88) prints 50 93 times

(i+89) prints 50 94 times

(i+90) prints 50 95 times

(i+91) prints 50 96 times

(i+92) prints 50 97 times

(i+93) prints 50 98 times

(i+94) prints 50 99 times

(i+95) prints 50 100 times

(i+96) prints 50 101 times

(i+97) prints 50 102 times

(i+98) prints 50 103 times

(i+99) prints 50 104 times

(i+100) prints 50 105 times

(i+101) prints 50 106 times

(i+102) prints 50 107 times

(i+103) prints 50 108 times

(i+104) prints 50 109 times

(i+105) prints 50 110 times

(i+106) prints 50 111 times

(i+107) prints 50 112 times

(i+108) prints 50 113 times

(i+109) prints 50 114 times

(i+110) prints 50 115 times

(i+111) prints 50 116 times

(i+112) prints 50 117 times

(i+113) prints 50 118 times

(i+114) prints 50 119 times

(i+115) prints 50 120 times

(i+116) prints 50 121 times

(i+117) prints 50 122 times

(i+118) prints 50 123 times

(i+119) prints 50 124 times

(i+120) prints 50 125 times

(i+121) prints 50 126 times

(i+122) prints 50 127 times

(i+123) prints 50 128 times

(i+124) prints 50 129 times

(i+125) prints 50 130 times

(i+126) prints 50 131 times

(i+127) prints 50 132 times

(i+128) prints 50 133 times

(i+129) prints 50 134 times

(i+130) prints 50 135 times

(i+131) prints 50 136 times

(i+132) prints 50 137 times

(i+133) prints 50 138 times

(i+134) prints 50 139 times

(i+135) prints 50 140 times

(i+136) prints 50 141 times

(i+137) prints 50 142 times

(i+138) prints 50 143 times

(i+139) prints 50 144 times

(i+140) prints 50 145 times

(i+141) prints 50 146 times

(i+142) prints 50 147 times

(i+143) prints 50 148 times

(i+144) prints 50 149 times

(i+145) prints 50 150 times

(i+146) prints 50 151 times

(i+147) prints 50 152 times

(i+148) prints 50 153 times

(i+149) prints 50 154 times

(i+150) prints 50 155 times

(i+151) prints 50 156 times

(i+152) prints 50 157 times

(i+153) prints 50 158 times

(i+154) prints 50 159 times

(i+155) prints 50 160 times

(i+156) prints 50 161 times

(i+157) prints 50 162 times

(i+158) prints 50 163 times

(i+159) prints 50 164 times

(i+160) prints 50 165 times

(i+161) prints 50 166 times

(i+162) prints 50 167 times

(i+163) prints 50 168 times

(i+164) prints 50 169 times

(i+165) prints 50 170 times

(i+166) prints 50 171 times

(i+167) prints 50 172 times

(i+168) prints 50 173 times

(i+169) prints 50 174 times

(i+170) prints 50 175 times

(i+171) prints 50 176 times

(i+172) prints 50 177 times

(i+173) prints 50 178 times

(i+174) prints 50 179 times

(i+175) prints 50 180 times

(i+176) prints 50 181 times

(i+177) prints 50 182 times

(i+178) prints 50 183 times

(i+179) prints 50 184 times

(i+180) prints 50 185 times

(i+181) prints 50 186 times

(i+182) prints 50 187 times

(i+183) prints 50 188 times

(i+184) prints 50 189 times

(i+185) prints 50 190 times

(i+186) prints 50 191 times

(i+187) prints 50 192 times

(i+188) prints 50 193 times

(i+189) prints 50 194 times

(i+190) prints 50 195 times

(i+191) prints 50 196 times

(i+192) prints 50 197 times

(i+193) prints 50 198 times

(i+194) prints 50 199 times

(i+195) prints 50 200 times

(i+196) prints 50 201 times

(i+197) prints 50 202 times

(i+198) prints 50 203 times

(i+199) prints 50 204 times

(i+200) prints 50

3) Pattern

```

    1 2
    2 2
  3 3 3
def pattern1(num):
  for i in range(1, num+1):
    for j in range(i):
      print(f'{j}{j}', end=' ')
    for k in range((num-i)+1):
      print(' ', end=' ')
    print("\n")
num = int(input("Enter the number\n"))
pattern1(num)

```

Output

3	1 2	2 2
2	2 2	3 3 3

4) Pattern

```

    1
    2 2
    3 2 3
def pattern2(num):
  for i in range(1, num+1):
    for j in range(i):
      print(f'{j}{j}', end=' ')
    for k in range((num-i)+1):
      print(' ', end=' ')
    print("\n")
num = int(input("Enter number\n"))
pattern2(num)

```

Output

1	2	3
---	---	---

4) Reverse of a number

```

num = int(input("Enter the number\n"))
reversed_num = 0
while num != 0:
  digit = num % 10
  reversed_num = reversed_num * 10 + digit
  num // 10
print("Reversed Number is: " + str(reversed_num))

```

Output

number : 4223	reversed number : 3224
---------------	------------------------

5) Program for bubble sort

```

def bubble_sort(list-item):
    for i in range(len(list-item)):
        for j in range(i+1, len(list-item)):
            if(list-item[i] > list-item[j]):
                temp = list-item[i]
                list-item[i] = list-item[j]
                list-item[j] = temp
    return list-item

```

~~for i in range(5):
list-item.append(int(input("Enter the element: ")))~~

K = int(input("Enter the size: "))
list-item = []
for i in range(K):
 list-item.append(int(input("Enter the element: ")))

list-item = bubble_sort(list-item)
print(list-item)

Output

Enter the size: 5

Enter the elements: 3 8 1

[1, 3, 8].

Lab - 3

```

list = [[], [], []]

for i in range(3):
    for j in range(3):
        list[i][j] = i+j+1

def create_board(list):
    for i in range(3):
        for j in range(3):
            print(list[i][j])
            print(" ")
        print(" ")
    print(" ")

def check_winner(list):
    for i in range(3):
        if (list[0][0] == list[1][1] == list[2][2]) or (list[0][0] == list[1][0] == list[2][0]):
            return True
        if (list[0][1] == list[1][1] == list[2][1]) or (list[0][1] == list[1][0] == list[2][0]):
            return True
        if (list[0][2] == list[1][2] == list[2][2]) or (list[0][2] == list[1][1] == list[2][0]):
            return True
    else:
        for j in range(3):
            y = list[0][j]
            x = list[0][0]
            count = 0, counts = 0
            if (i == j) & (list[i][j] == x):
                count++
            if (i + j == 2) & (list[i][j] == x):
                counts++

```

```
if (count == 3 || count == 3)
    return true.
```

Algorithm

```
def play(k,j)
    n=k/3
    if (list[n][k%3] == 'x' || list[n][k%3] == 'o')
        print("player", " has already played this position")
    else :
        if (j==1):
            list[n][k%3] = 'x'
        else:
            list[n][k%3] = 'o'
```

Algorithm

```
def create_board():
    list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

1) Fill the list with dummy numbers.
2) Write create_board function → helps in pointing board.

3) Create a function which runs after a player plays and checks for the winning conditions.

4) Create a function play function which helps players to switch their chances to play.

5) Main function takes input from the players and calls the above functions.

Count=0

while(1):

Output

24/11/20

```
create_board(list)
print("player", "count", "to play")
x=int(input())
play(x, count%2+1)

if (check_winner()):
    print("player", "won")
    break
    count+=1
```

Lab 4

PAGE NO: _____
DATE: _____

PAGE NO: _____
DATE: _____

Code

```
import copy
from heap import heappush, heappop.
```

```
n = 3.
now = [1, 0, -1, 0]
to1 = [0, -1, 0, 1]
```

```
class PriorityQueue:
    def __init__(self):
        self.heap = []
    def push(self, k):
        heappush(self.heap, k)
    def pop(self):
        return heappop(self.heap)
```

```
def empty(self):
    return not bool(self.heap)
```

class Node:

```
def __init__(self, parent, notEmptyHeap,
             level):
    self.parent = parent
```

```
self.empty_file_pos = empty_file_pos
self.level = level
self.notEmpty = notEmpty
```

```
def __lt__(self, next):
    return self.level <= next.level
```

```
def __eq__(self, next):
    if self.level == next.level:
        if self.empty_file_pos == next.empty_file_pos:
            if self.notEmpty == next.notEmpty:
                return True
    return False
```

(P)
2/11

Algorithm

- 1) Create a **Puzzle** class and define initial configuration & generate the numbers to generate a random initial board.
- 2) Point the current Board & implement a method for it to point.
- 3) For moving of tiles \rightarrow Implement a method to tiles based on ~~BFS~~ ~~using insert & check for~~ validity of the moves. to avoid a method to check if the current board configuration matches the goal state.
- 4) Name loop:
Create a game loop that continues till puzzle is solved.
In each iteration of the loop:
print the current state of board.
prompt the user for a move direction.
Move the tiles accordingly.
Check if the puzzle is solved.
If solved, break out of the loop.
If not solved, repeat the loop.

```

def calculateCost(mat, final):
    count = 0
    for i in range(n):
        for j in range(n):
            if mat[i][j] and mat[i][j] == final[i][j]:
                find(i, j)
                count += 1
    return count

```

```

def newNode(mat, empty_tile_pos, new_empty_tile_pos, parent, final):
    new_mat = np.copy(mat)

```

```

    X1, Y1 = empty_tile_pos
    X2, Y2 = new_empty_tile_pos
    new_mat[X1][Y1], new_mat[X2][Y2] = new_mat[X2][Y2], new_mat[X1][Y1]
    cost = calculateCost(new_mat, final)
    return newNode()

```

Solve()

J

On

1	2	3
4	0	6
7	5	8

1	2	3
4	5	6
7	8	0

Output:

```

initial          2   8
1   3           0   4
8   0   4       2   6
7   6   5       5

```

Output

1	2	3
0	4	6
7	5	8

Lab 5

Page No:
Date:

$$f = g \frac{d}{dx} \rightarrow$$

class Node:
, dpt -- (self.data, level, parent)
self.data = data
self.

q = 1, h = 3, p = 3

q=0, h=3, p=3

q=1, h=4, p=3

q=2

q=3

q=4

q=5

q=6

q=7

q=8

q=9

q=10

q=11

q=12

q=13

q=14

q=15

q=16

q=17

q=18

q=19

q=20

q=21

q=22

q=23

q=24

q=25

q=26

q=27

q=28

q=29

q=30

q=31

q=32

q=33

q=34

q=35

q=36

q=37

q=38

q=39

q=40

q=41

q=42

q=43

q=44

q=45

q=46

q=47

q=48

q=49

q=50

q=51

q=52

q=53

q=54

q=55

q=56

q=57

q=58

q=59

q=60

q=61

q=62

q=63

q=64

q=65

q=66

q=67

q=68

q=69

q=70

q=71

q=72

q=73

q=74

q=75

q=76

q=77

q=78

q=79

q=80

q=81

q=82

q=83

q=84

q=85

q=86

q=87

q=88

q=89

q=90

q=91

q=92

q=93

q=94

q=95

q=96

q=97

q=98

q=99

q=100

q=101

q=102

q=103

q=104

q=105

q=106

q=107

q=108

q=109

q=110

q=111

q=112

q=113

q=114

q=115

q=116

q=117

q=118

q=119

q=120

q=121

q=122

q=123

q=124

q=125

q=126

q=127

q=128

q=129

q=130

q=131

q=132

q=133

q=134

q=135

q=136

q=137

q=138

q=139

q=140

q=141

q=142

q=143

q=144

q=145

q=146

q=147

q=148

q=149

q=150

q=151

q=152

q=153

q=154

q=155

q=156

q=157

q=158

q=159

q=160

q=161

q=162

q=163

q=164

q=165

q=166

q=167

q=168

q=169

q=170

q=171

q=172

q=173

q=174

q=175

q=176

q=177

q=178

q=179

q=180

q=181

q=182

q=183

q=184

q=185

q=186

q=187

q=188

q=189

q=190

q=191

q=192

q=193

q=194

q=195

q=196

q=197

q=198

q=199

q=200

q=201

q=202

q=203

q=204

q=205

q=206

q=207

q=208

q=209

q=210

q=211

q=212

q=213

q=214

q=215

q=216

q=217

q=218

q=219

q=220

q=221

q=222

q=223

q=224

q=225

q=226

q=227

q=228

q=229

q=230

q=231

q=232

q=233

q=234

q=235

q=236

q=237

q=238

q=239

q=240

q=241

q=242

q=243

q=244

q=245

q=246

q=247

q=248

q=249

q=250

q=251

q=252

q=253

q=254

q=255

q=256

q=257

q=258

q=259

q=260

q=261

q=262

q=263

q=264

q=265

q=266

q=267

q=268

q=269

```

def smallt(SLP, pu2, x1, y2, pu1, y1):
    if x1 >= 0 & x2 <= 1 & SLP.data == 0 & y2 >= 0 &
       y1 <= len(SLP.data):
        temp = []
        temp.append(pu2)
        temp.append(pu1)
        return temp
    else:
        return None

```

```

def h(SLP, goal):
    "calculates the distance of missing values"
    if SLP == 0:
        for i in range(0, len(SLP)):
            for j in range(0, SLP[i].n):
                if SLP[i][j] != goal[i][j] and
                   str(i) + str(j) != '_':
                    temp += 1
    return temp

```

```

def process(SLP):
    return temp.

```

```

start = SLP.accept()
goal = SLP.accept()

```

```

start = Node((start, 0, 0))
start.goal = SLP.f(goal, goal)
SLP.open.append(start)
while True:
    com = SLP.pop(0)

```

```

for i in com.data:

```

```

    for j in i:

```

```

        point(j, end=" ")
        print(" ")

```

```

if (SLP.h(com, data, goal) == 0):
    if (SLP.h(com, data, goal) == 0):
        bocat =
        return SLP.h(com, data, goal) + SLP.h(bocat, goal)
    else:
        accept(SLP):
        "Accept process from user"
        def f(SLP, start, goal):
            return SLP.h(com, data, goal) + SLP.h(goal, bocat)

```

```

# for i in com.generate_child():
    i.fval = SLP.f(i, goal)
    SLP.open.append(i)
    SLP.closed.append(i)
    del SLP.open[0]

```

TDPS.

`set1, open = (key:lambda x: x[1], val,
to_visit=False)`

```
pu2 = pu2.visit(3)
pu2.apopen()
```

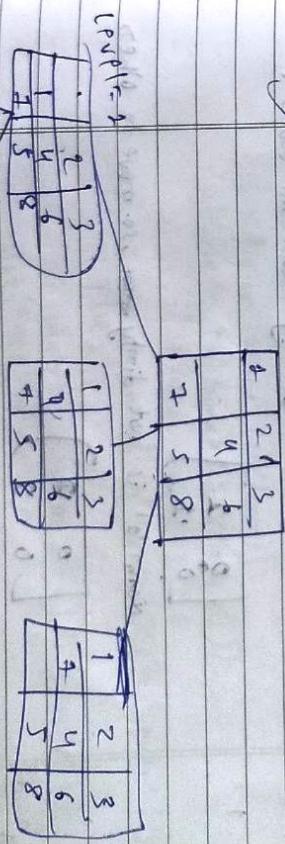
Output

```
def dts(pu2, goal, get_moves):
    if depth == 0:
        return []
    if route[-1] == goal:
        return route
    for move in get_moves(route[-1]):
        if move not in route:
            next_route = dts(pu2, goal, depth - 1)
            if len(next_route) > 0:
                if move in next_route:
                    return next_route
    return route
```

```
for depth in range(0, count):
    route = dts(pu2, goal, depth)
```

if route:

return route



Code

```
def clean(floor):
    for row in range(len(floor)):
        for i in range(len(floor[0])):
            if floor[i] == 0:
                print("Floor[i] = 0.
```

use:

```
for j in range(col-2, -1, -1):
    if floor[j] == 2:
        point_f(floor[i][j])
```

```
point_f(floor[i][j]) = 0.
```

Step 1: Create room array.
Step 2: In each cell, if room is dirty mark it as '1' and if room is 'clean' mark it as '0'.
Step 3: Create a clean function through which the cleaner moves right, left, up, down & checks the room is dirty or not.
Step 4: After cleaning the room travel to next room & stops no cleaning.

Eg:

```
[0 1]
```

Room '1' is not dirty, so move to others

```
[0 1]
```

Room '2' is dirty clean the room.

```
print_f(floor, j, i)
```

```
def point_f(floor, row, col):
    point("The floor matrix is as below:")
    for r in range(len(floor)):
        for c in range(len(floor[r])):
            if r==row and c==col:
                print("Floor[" + str(r) + "][" + str(c) + "] = 1")
            else:
                print("Floor[" + str(r) + "][" + str(c) + "] = 0")
```

Room '2' is not dirty, so move to others

```
[0 0]
```

Printed

```
printf("%d", floor[r][c], end=1)
print("end=\n")
print(end=\n)
```

Knowledge Lab

PAGE NO:
DATE:

29/12/23

R
PAGE NO:
DATE:

def main()

floor = []

m = input

n = input

print("Enter the clean rooms")

for i in range(m):

p = list(map(int, input().split(" ")))

floor.append(p)

print(floor)

clean([0,0])

now()

out()

water

wet

ground

plants

grow

rain

snow

ice

sun

clouds

rainy

sunny

cloudy

overcast

rainy

sunny

cloudy

overcast

rainy

sunny

S. Jitendra

knowledge base embodiment

It is raining (P), then ground is wet (Q)

If the ground is wet (Q), then the plants will grow (R).

It's not the case that plants will grow (~R).

Query - Whether it is raining.

Code

from sympy import symbols, And, Not, Implies, satisfiable

def create-knowledge-base():

P=symbols('P')

Q=symbols('Q')

R=symbols('R')

knowledge_base = And(

Implies(P,Q),

Implies(Q,R),

Not(R))

)

return knowledge_base

Result 0 is dirty.

0 > 0c

1 0.

def query_kb(knowledge_base, query);

entailment = satisfiable(And(knowledge-base,
Not(query))))

return not entailment.

def add_fact(self, fact):
 self.facts.append(fact)

if name == "main":

kb = create_knowledge_base()

query = symbol('p')

result = query_kb(kb, query)

def rule_0(statement):
 return 'p' in statement.lower()

def rule_1(statement):
 return "not" in statement.lower() or "or"
 "and" in statement.lower()

in statement.lower()

def rule_2(statement):

def rule_3(statement):
 return "not" in statement.lower()
 or "or" in statement.lower()
 or "and" in statement.lower()

Log 9

PAGE NO.:
DATE:

Implement unification in FOL

KB = Knowledge Base
KB is a fact (null = 0)

```
def getStatement = input("Enter a statement: ")  
usem_statement = KB + " " + usem_statement  
if len(usem_statement) > 0:  
    KB += usem_statement
```

if entailResult:
 print("The statement is entailed by KB")

print("The statement is not entailed by KB")

```
def getAttributes(expression):  
    expression = expression.split("(")[1:-1]  
    expression = expression[:-1] + expression[-1]  
    expression = re.split("(?<=[\.\(\)]), (?![\.\)])", expression)  
    return expression
```

```
def getInitialPredicate(expression):  
    return expression.split("(")[0]
```

```
def isVariable(cha):  
    return char.islower() and len(char) == 1
```

```
def apply(pp, substitutions):  
    for substitution in substitutions:  
        new, old = substitution  
        exp = replaceAttributes(pp, old, new)
```

for substitution in substitutions:
 new, old = substitution
 exp = replaceAttributes(pp, old, new)

Notes

usem

Entailment

The statement is not entailed by the KB.

Notes

(exp) is a predicate, (exp) is a variable
Entailment is a relation between two statements
KB is a knowledge base
Null value

```

def unify(exp1, exp2):
    if exp1 == exp2:
        return []
    if isConstant(exp1) and isConstant(exp2):
        if attributeCount1 != attributeCount2:
            return False
        if attributeCount1 == attributeCount2:
            return [exp1, exp2]
    if isConstant(exp1):
        return [exp1, exp2]
    if isVariable(exp1):
        return [exp1, exp2]
    if isVariable(exp2):
        return [exp2, exp1]
    if checkOccurs(exp1, exp2):
        return False
    else:
        return [exp1, exp2]
    if isVariable(exp2):
        if checkOccurs(exp2, exp1):
            return False
        else:
            return [exp1, exp2]
    if isVariable(exp1):
        if checkOccurs(exp1, exp2):
            return False
        else:
            return [exp1, exp2]
    if attributeCount1 != attributeCount2:
        return False
    if attributeCount1 == attributeCount2:
        heads = getFirstPoint(exp1)
        heads2 = getNextPoint(exp2)
        initialSubstitution = unify(heads, heads2)
        if not initialSubstitution:
            return False
        fail1 = getRemainingPoint(exp1)
        fail2 = getRemainingPoint(exp2)
        if fail1 == fail2:
            return initialSubstitution
        failing:
            if initialSubstitution != []:
                fail1 = apply(fail1, initialSubstitution)
                fail2 = apply(fail2, initialSubstitution)
            if failingSubstitution == unify(fail1, fail2):
                if not failingSubstitution:
                    return failingSubstitution
            initialSubstitution.extend(failingSubstitution)
            return initialSubstitution
    if getInitialPredicate(exp1) != getInitialPredicate(exp2):
        print("Predicates do not match. Cannot be unified")
        return False

```

Part Program 2 m

$\text{exp}_2 = \lambda \text{ knows}(\text{A}, x)$
 $\text{exp}_2 = " \text{knows}(\text{mother}('y'))"$

```
def getAttributedString:  
    expr = (([n]) + r) i
```

Output → point (isubstitution)

```
def getPredicates(string):
```

... (A) \vdash $\neg p \vee q$ (from 1)

$\text{B}_{\text{P}_2} - \text{B}_{\text{N}_2}$ (y) =
substitutions = unity ($\text{C}_{\text{P}_2}, \text{C}_{\text{N}_2}$)

fridge

~~$(C(A), x), (B, y)$~~

def SHOLMINIZATION (Convenience)
SYNTH-CONSTANTS = $\Gamma \vdash y : \text{for}(\alpha) y'$

~~Statement = join list (Sentence copy(1))
maths = ex. R. Radoul (1 \times [C^n]) + J,~~

جذب

Statement = statement · variable (3, 3, 2)
for predicate in interpretation (statement)
attributes = generalizations (~~statements~~)
predicate

PAGE NO :
DATE :

PAGE NO :
DATE :

DATE
PAGE NO.

PAGE NO :
DATE :

11) Word create : a knowledge consisting of best

old logic statements & prove them given

terminal posterior furcula frank

import ~~src~~: Jupyter E ?

```
def isVariable(x):    # Checks whether the variable
```

x.isalpha()

```
def getAttributes(string):  
    expr = r'\{(\w+)\}\n';
```

```
matches = re.findall(rps, string)
```

def getPredicates(string):

$\text{expr} = ((a - 2n) +) \backslash ((\text{Eng} (+))$
return $\text{xL}.$ findall(expr , Sizing)

100

`apt-init - (soft, permission), (0x2) flags`

~~predicates, operators~~ = ~~statements~~ = statements

What is the value of $(1) \times (2) \times (3) \times (4)$?

App::splitExpression(300, \$expression);

`predicate = getPredicate(expression)`

11 used to split expression.

```
def getResult(splf):
```

— 1 —

from given expression.

schon (v. if. is Vomöbeln) else name like vint

Sept. params).

class Implication:

def evaluate(self, tasks): This makes it the
autorenders are

*new-
hs=[]
satisfied by the
fact infants: facts in the KB.*

for val in sorted(his:

for i, v in enumerate (val. of Variable)

$kb = KB()$

$kb.tell('king(x) \& greedy(x) \Rightarrow evil(x)')$

$kb.tell('King(John)')$

$kb.tell('greedy(John)')$

$kb.tell('king(Richard)')$

$kb.greedy('evil(x)')$

Output:

Querying 'evil(x)'

1. evil(Richard)

2. evil(John)

Signature
24/1/19

```
class KB:
    def __init__(self):
        self.facts = set()
        self.implicit = set()

    def tell(self, p):
        if (p[0] == 'if') in self.implicit:
            self.implicit.add(p)
        else:
            self.facts.add(p)

    def implicationadd(self, Implication):
        self.implicit.add(Implication)

def query(self, p):
    for f in self.facts:
        if fact(f).predicate == fact(p).predicate:
            print(f'if {f} then {p}'')
            return
    print('not found')

def display(self):
    for i, f in enumerate(self.facts):
        print(f'{i+1}. {f}'')
```

```
kb = KB()
kb.tell('missile(x) \Rightarrow weapon(x)')
kb.tell('missile(ma)')  

kb.tell('enemy(r, America) \Rightarrow hostile(r)')  

kb.tell('American(West)')  

kb.tell('owns')  

kb.tell('if missile(x) then weapon(x)')  

kb.tell('if missile(ma)')  

kb.tell('if enemy(r, America) then hostile(r)')  

kb.tell('if American(West)')  

kb.tell('owns')
```