**Questions:**

1. **Define a recursive procedure in Python and in Prolog to find the sum of 1st n terms of an equal-interval series given the 1st term and the interval.**
2. **Define a recursive procedure in Python and in Prolog to find the length of a path between two vertices of a directed weighted graph.**
3. **Modify the Python and Prolog codes demonstrated above to find h2 and h3 discussed above.**

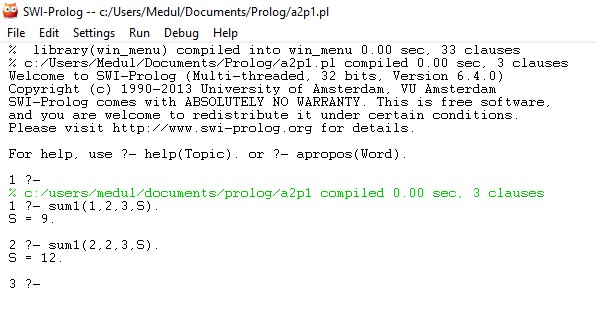
**Solution to the question no. 1:**

**The demonstrated Prolog code to find the sum of 1st n terms of an equal-interval series is as below:**

sum1(F,\_,1,F):-!.

sum1(F,I,N,S):-N1 is N-1, sum1(F, I, N1, S1), S is S1+F+(N-1)\*I.

**A sample input and output is as below:**



**The demonstrated Python code to find the sum of 1st n terms of an equal-interval series is as below:**

def calSum(N,I,F):

if (N==0):

return 0

elif (N>=1):

return calSum(N-1,I,F)+F+(N-1)\*I

# Main

t=int(input('How many times?\n'))

for i in range(t):

print('Iteration:',i+1)

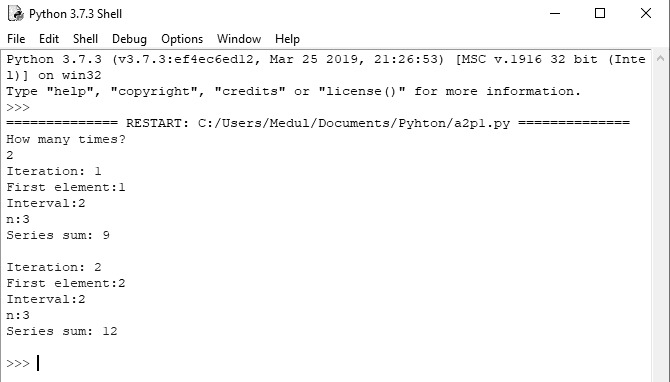
f=int(input('First element:'))

d=int(input('Interval:'))

n=int(input('n:'))

print('Series sum:', calSum(n,d,f),'\n')

**A sample input and output is as below:**



**Solution to the question no. 2:**

**The demonstrated Prolog code to find length of a path between two vertices of a directed weighted graph is as below:**

neighbor(i,a,35). neighbor(i,b,45). neighbor(a,c,22).

neighbor(a,d,32). neighbor(b,d,28). neighbor(b,e,36).

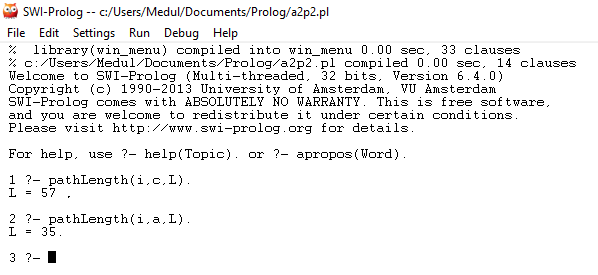
neighbor(b,f,27). neighbor(c,d,31). neighbor(c,g,47).

neighbor(d,g,30). neighbor(e,g,26).

pathLength(X,Y,L):- neighbor(X,Y,L),!.

pathLength(X,Y,L):- neighbor(X,Z,L1), pathLength(Z,Y,L2), L is L1+L2.

**A sample input and output is as below:**



**The demonstrated Python code to find length of a path between two vertices of a directed weighted graph is as below:**

neighbor = {

'i':[('a',35),('b',45)],

'a':[('c',22),('d',32)],

'b':[('d',28),('e',36),('f',27)],

'c':[('d',31),('g',47)],

'd':[('g',30)],

'e':[('g',27)]

}

def find\_path\_length(neighbor, start, end, path =[], cost=0):

path = path + neighbor[start]

if start == end:

return cost

for (node,weight) in neighbor[start]:

if node not in path:

return find\_path\_length(neighbor, node, end, path, cost+weight)

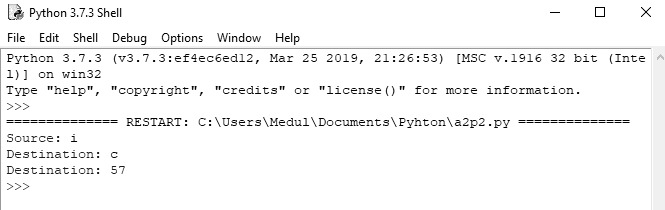
#Main

s = input('Source: ')

d = input('Destination: ')

print('Destination: '+str( find\_path\_length(neighbor, s, d) ))

**A sample input and output is as below:**



**Solution to the question no. 3(a) :**

**The demonstrated Prolog code to find h2 (Manhattan distances of the tiles) is as below:**

gtp(1,1,1). gtp(2,1,2). gtp(3,1,3). gtp(4,2,3). gtp(5,3,3). gtp(6,3,2). gtp(7,3,1). gtp(8,2,1). gblnk(2,2).

tp(1,1,2). tp(2,1,3). tp(3,2,1). tp(4,2,3). tp(5,3,3). tp(6,2,2). tp(7,3,2). tp(8,1,1). blnk(3,1).

go:- calcH(1,[],L), sumList(L,V),write('Manhattan distance: '),write(V).

calcH(9,X,X):-!.

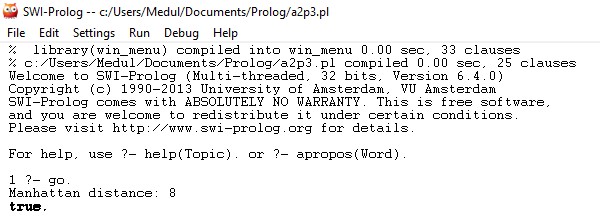
calcH(T,X,Y):- dist(T,D), append(X,[D],X1), T1 is T+1, calcH(T1,X1,Y).

dist(T,V):-tp(T,A,B), gtp(T,C,D), V is abs(A-C) + abs(B-D).

sumList([],0):-!.

sumList(L,V):-L=[H|T], sumList(T,V1), V is V1+H.

**A sample input and output is as below:**



**The demonstrated Python code to find h2 (Manhattan distances of the tiles) is as below:**

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

gblnk = (2,2)

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

blnk = (3,1)

def manhattan():

cost = 0

for i in range(8):

cost += abs(gtp[i][1] - tp[i][1]) + abs(gtp[i][2] - tp[i][2])

return cost

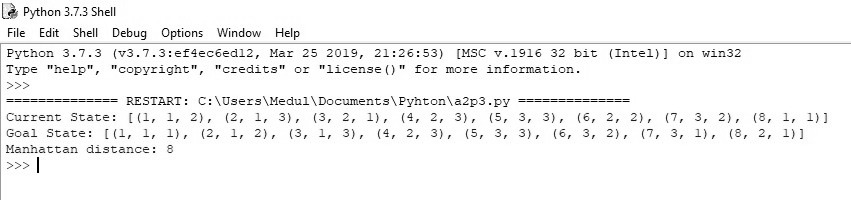
#Main

print('Current State: '+str(tp))

print('Goal State: '+str(gtp))

print('Manhattan distance: '+str( manhattan() ))

**A sample input and output is as below:**



**Solution to the question no. 3(b):**

**The demonstrated Prolog code to find h3 (number of attacking pairs of queens in 8-queens problem) is as below:**

:-dynamic(hval/1).

/\* Evaluates a 8-queens' state given as list of 8 digits \*/

evalState(L,V):- assert(hval(0)),hl(1,L), di\_up(1,L),di\_dn(1,L),hval(V),

retractall(hval(\_)).

hl(8,\_):-!.

hl(I,L):- nthel(I,L,X), chk\_incr(I,L,X), I1 is I+1, hl(I1,L).

chk\_incr(8,\_,\_):-!.

chk\_incr(I,L,X):- I1 is I+1, nthel(I1,L,Y),

do\_incr(X,Y),chk\_incr(I1,L,X).

do\_incr(X,Y):- X=Y, incr\_hval.

do\_incr(\_,\_).

incr\_hval:-hval(V), V1 is V+1, retract(hval(\_)), assert(hval(V1)).

di\_up(8,\_):-!.

di\_up(I,L):- nthel(I,L,X), chkup\_incr(I,L,X,0), I1 is I+1,

di\_up(I1,L).

chkup\_incr(8,\_,\_,\_):-!.

chkup\_incr(I,L,X,K):- I1 is I+1, nthel(I1,L,Y), K1 is K+1,

doup\_incr(X,Y,K1), chkup\_incr(I1,L,X,K1).

doup\_incr(X,Y,K1):- X1 is X+K1, Y=X1, incr\_hval.

doup\_incr(\_,\_,\_).

di\_dn(8,\_):-!.

di\_dn(I,L):- nthel(I,L,X), chkdn\_incr(I,L,X,0), I1 is I+1,

di\_dn(I1,L).

chkdn\_incr(8,\_,\_,\_):-!.

chkdn\_incr(I,L,X,K):- I1 is I+1, nthel(I1,L,Y), K1 is K+1,

dodn\_incr(X,Y,K1), chkdn\_incr(I1,L,X,K1).

dodn\_incr(X,Y,K1):- X1 is X-K1, Y=X1, incr\_hval.

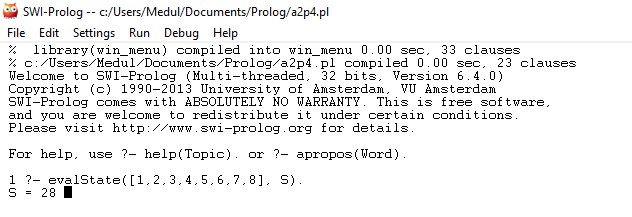
dodn\_incr(\_,\_,\_).

% A procedure to find the nth element of a list

nthel(N,[\_|T],El):- N1 is N-1, nthel(N1,T,El).

nthel(1,[H|\_],H):-!.

**A sample input and output is as below:**



**The demonstrated Python code to find h3 (number of attacking pairs of queens in 8-queens problem) is as below:**

qList = [1,2,3,4,5,6,7,8]

def evalState(qList):

count = 0

#horizontal check

for i in range(0,len(qList),1):

for j in range(i+1,len(qList),1):

if (qList[i] == qList[j]):

count = count + 1

#diagonal\_up check

for i in range(0,len(qList),1):

x = qList[i]

for j in range(i+1,len(qList),1):

x = x + 1

if (x == qList[j]):

count = count + 1

#diagonal\_down check

for i in range(0,len(qList),1):

x = qList[i]

for j in range(i+1,len(qList),1):

x = x - 1

if (x == qList[j]):

print(str(i)+" "+str(j))

count = count + 1

return count

#Main

print('Queen Position: '+str(qList))

print('Total Attacking Pairs: '+str( evalState(qList) ))

**A sample input and output is as below:**

