SIES (Nerul) College of Arts, Science & Commerce

RISE WITH EDUCATION

SIES (NERUL) COLLEGE OF ARTS, SCIENCE AND COMMERCE

Navi Mumbai - 400706

JOURNAL

FOR THE SUBJECT OF

ARTIFICIAL INTELLIGENCE PRACTICAL

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE AWARD OF THE DEGREE OF B.SC. (INFORMATION TECHNOLOGY)

SUBMITTED BY

Sylvia John Basil

T.18.05

Table of Contents

- Uninformed - Domain not known only present start node known.
- Deeper node (one direct)
- Stack (LIFO)
- There's chance of not getting goal state - or could get stuck in loop

PRACTICAL NO.-1

Write a program to implement depth first search algorithm.

AIM:

Write a program to implement depth first search algorithm.

ACF BED
ACF BDE

(ABEFCD (coming to stack 2 times))

GRAPH:

A

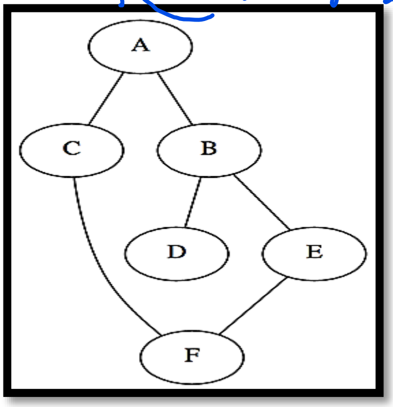
AC (when remove node, add children push into stack)

ACF

ACFB (add child of E)

ACFBE

ACFBED



PYTHON CODE:

```
# sample graph implemented as a dictionary graph1
= {
        'A': set(['B', 'C']),
        'B': set(['A', 'D', 'E']),
        'C': set(['A', 'F']),
        'D': set(['B']),
        'E': set(['B','F']),
        'F': set(['C','E'])
       }
def dfs(graph, node, visited):
if node not in visited:
visited.append(node)
for n in graph[node]:
dfs(graph,n,visited)     return
visited visited =
dfs(graph1,'A', [])
print(visited)
```

node = key n = value of key

(instead of printing we return val, coz of recursion

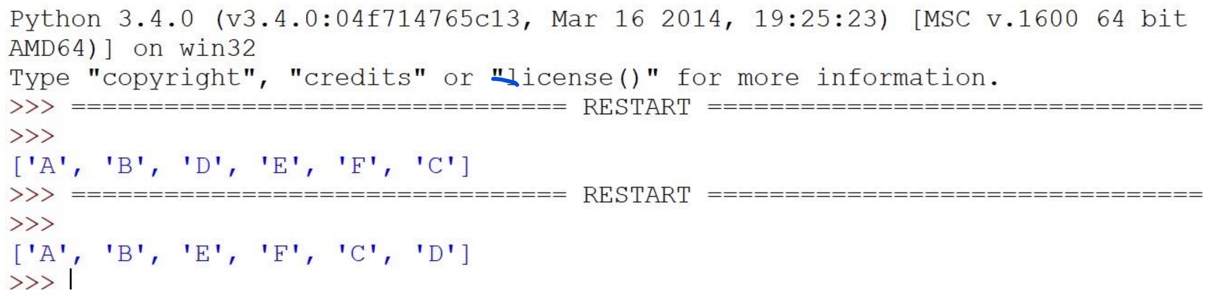
returned

empty list sending for visited

stored val

we are calling this funct for every n value.

OUTPUT:



```
Python 3.4.0 (v3.4.0:04f714765c13, Mar 16 2014, 19:25:23) [MSC v.1600 64 bit
AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
['A', 'B', 'D', 'E', 'F', 'C']
>>> ================================ RESTART ================================
>>>
['A', 'B', 'E', 'F', 'C', 'D']
>>> |
```

App - to traverse
- to find a node / element
- to find path
- problem solving.
- gives shortest path. in shortest time
- to cover all nodes.

PRACTICAL NO.-2

Write a program to implement breadth first search algorithm

to check
- traversal order
- to check path (A-C-F)
- will have multiple ans.
- uninfored/Blind search
  - no Domain Spec knowledge
  - no prior knowledg of how to get to goal
  - only present node (start node known)
- Follows FIFO (queue)
- level-order search
- doesn't leave any node behind, while searching
- optimal - gives shortest result.

AIM:

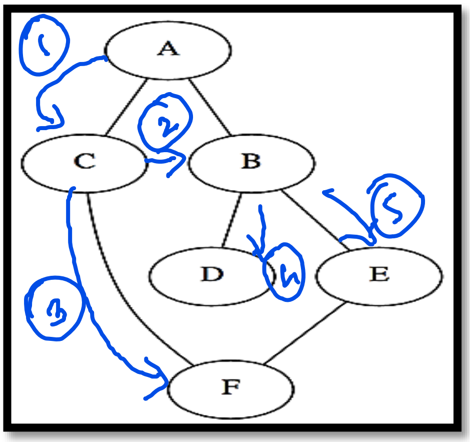
Write a program to implement breadth first search algorithm.

GRAPH:

neigh

ACB, neig
A C B F
① A C B F D E
② A C B F E D

A BC
ABCDE
ABCDEF
ABCEDF



PYTHON CODE:

# sample graph implemented as a dictionary

```
graph1 = {
 'A': set(['C', 'B']),
 'B': set(['A', 'D', 'E']),
 'C': set(['A', 'F']),
 'D': set(['B']),
 'E': set(['B', 'F']),
 'F': set(['C', 'E'])
 } visited = [] # List to keep track of visited
nodes.
queue = [] #Initialize a queue
def bfs(visited, graph, node):
queue.append(node)
visited.append(node)      while
queue:
          s = queue.pop(0) #A          print(s, end = " ")
for neighbour in graph[s]: #dict[key]-->graph['A']
if neighbour not in visited:
queue.append(neighbour)
visited.append(neighbour)
bfs(visited, graph1, 'A')
```

[C, B] - list neighbours of A

→ starting node

add

add to queue & mark as marked until queue empty

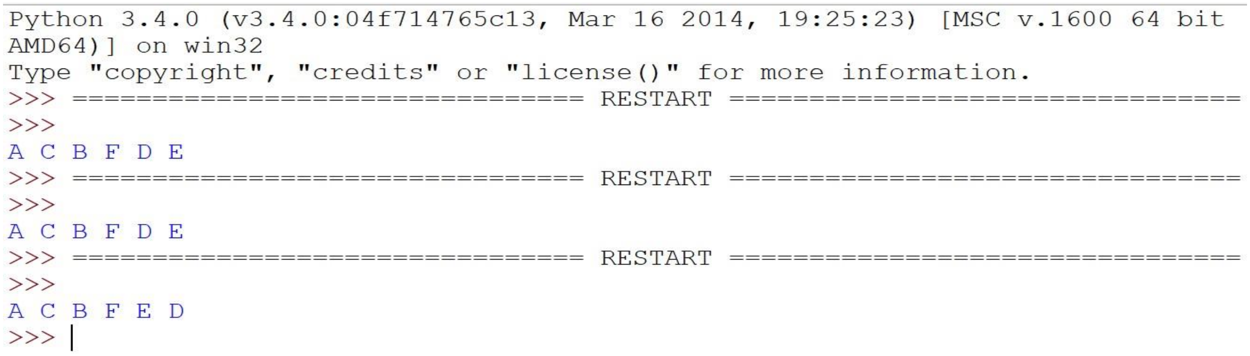
first item in queue is in 0th position

→ all elements get printed on same line together.

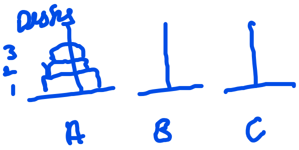
→ C, B

then it'll check entire graph

OUTPUT:



```
Python 3.4.0 (v3.4.0:04f714765c13, Mar 16 2014, 19:25:23) [MSC v.1600 64 bit
AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>> 
A C B F D E
>>> ================================ RESTART ================================
>>> 
A C B F D E
>>> ================================ RESTART ================================
>>> 
A C B F E D
>>> |
```



(Source) ↓ (target

Intermediate pole u can use

2 conditions to play/solve

① at a time, u can move only 1 disk

② when u keep a disk on top of another, u can keep small on top of big but not big on small

→Recursion - to repeat. to call function inside same function or func can call itself.

PRACTICAL NO.-3

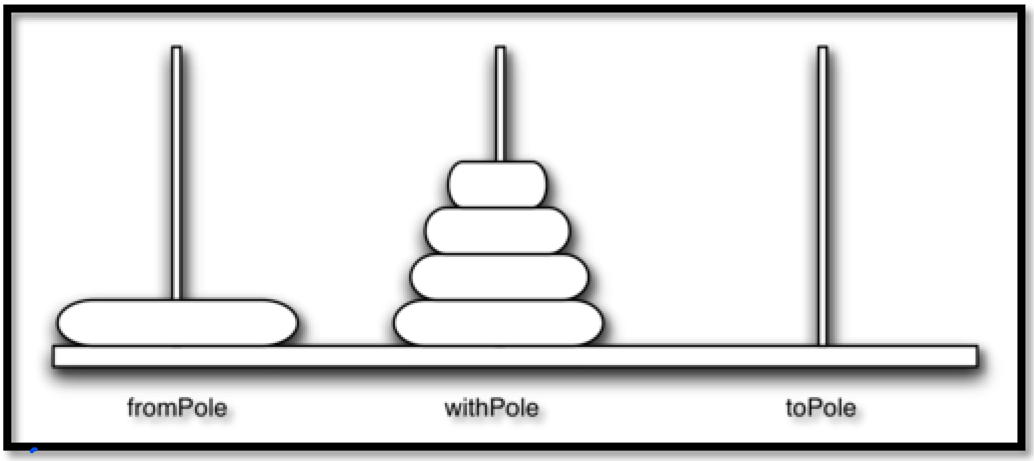
A→C
A→B
C→B
A→C
B→A
B→C
A→C

Write a program to solve tower of Hanoi problem.

AIM:

Write a program to solve tower of Hanoi problem.

DIAGRAM:



recursion based program

PYTHON CODE:

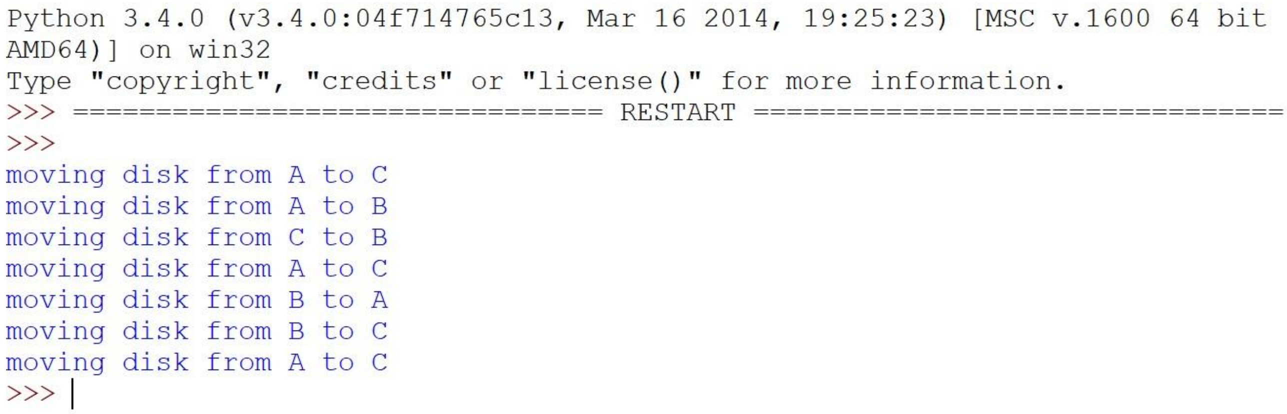
func →total no of poles

```
def moveTower(height,fromPole, toPole, withPole):
if height >= 1:
    moveTower(height-1,fromPole,withPole,toPole)
moveDisk(fromPole,toPole)      moveTower(height-
1,withPole,toPole,fromPole) def moveDisk(fp,tp):
     print("moving disk from",fp,"to",tp) moveTower(3,"A","C","B")
```

will keep changing

So transfer will happen only between 2 poles from which pole to which you want to move

OUTPUT:



```
Python 3.4.0 (v3.4.0:04f714765c13, Mar 16 2014, 19:25:23) [MSC v.1600 64 bit
AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
moving disk from A to C
moving disk from A to B
moving disk from C to B
moving disk from A to C
moving disk from B to A
moving disk from B to C
moving disk from A to C
>>> |
```

PRACTICAL NO.-4

Write a program to implement A* algorithm.

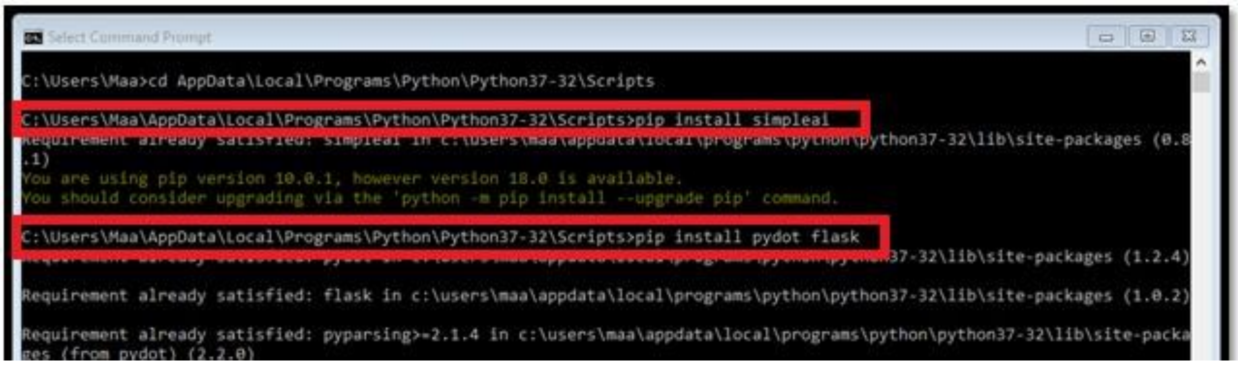
AIM:

Write a program to implement A* algorithm.

NOTE:

Install 2 packages in python scripts directory using pip command.

1. **pip install simpleai**
2. **pip install pydot flask**



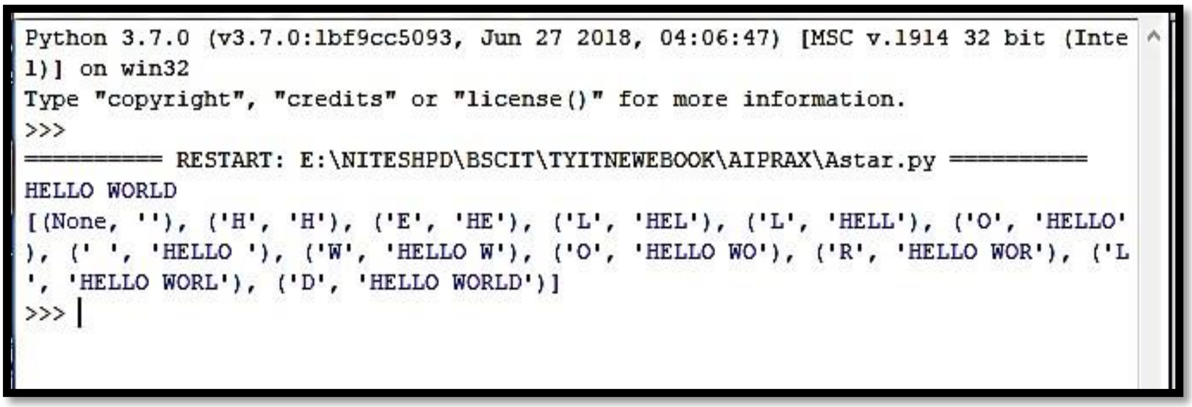
```
C:\Users\Maa>cd AppData\Local\Programs\Python\Python37-32\Scripts

C:\Users\Maa\AppData\Local\Programs\Python\Python37-32\Scripts>pip install simpleai
Requirement already satisfied: simpleai in c:\users\maa\appdata\local\programs\python\python37-32\lib\site-packages (0.8
.1)
You are using pip version 10.0.1, however version 18.0 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

C:\Users\Maa\AppData\Local\Programs\Python\Python37-32\Scripts>pip install pydot flask
Requirement already satisfied: pydot in c:\users\maa\appdata\local\programs\python\python37-32\lib\site-packages (1.2.4)

Requirement already satisfied: flask in c:\users\maa\appdata\local\programs\python\python37-32\lib\site-packages (1.0.2)

Requirement already satisfied: pyparsing>=2.1.4 in c:\users\maa\appdata\local\programs\python\python37-32\lib\site-packa
ges (from pydot) (2.2.0)
```

PYTHON CODE:

```
from simpleai.search import SearchProblem, astar
GOAL = 'HELLO WORLD'
class HelloProblem(SearchProblem):
def actions(self, state):         if
len(state) < len(GOAL):
        return list(' ABCDEFGHIJKLMNOPQRSTUVWXYZ')
else:
```

```
            return []     def
result(self, state, action):
        return state + action
def is_goal(self, state):
return state == GOAL     def
heuristic(self, state):
        # how far are we from the goal?
wrong = sum([1 if state[i] != GOAL[i] else 0
for i in range(len(state))])         missing =
len(GOAL) - len(state)         return wrong +
missing problem = HelloProblem(initial_state='')
result = astar(problem) print(result.state)
print(result.path())
```

OUTPUT:



```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: E:\NITESHPD\BSCIT\TYITNEWEBOOK\AIPRAX\Astar.py ==========
HELLO WORLD
[(None, ''), ('H', 'H'), ('E', 'HE'), ('L', 'HEL'), ('L', 'HELL'), ('O', 'HELLO'
), (' ', 'HELLO '), ('W', 'HELLO W'), ('O', 'HELLO WO'), ('R', 'HELLO WOR'), ('L
', 'HELLO WORL'), ('D', 'HELLO WORLD')]
>>> |
```

PRACTICAL NO.-5

Write a program to solve water jug problem.

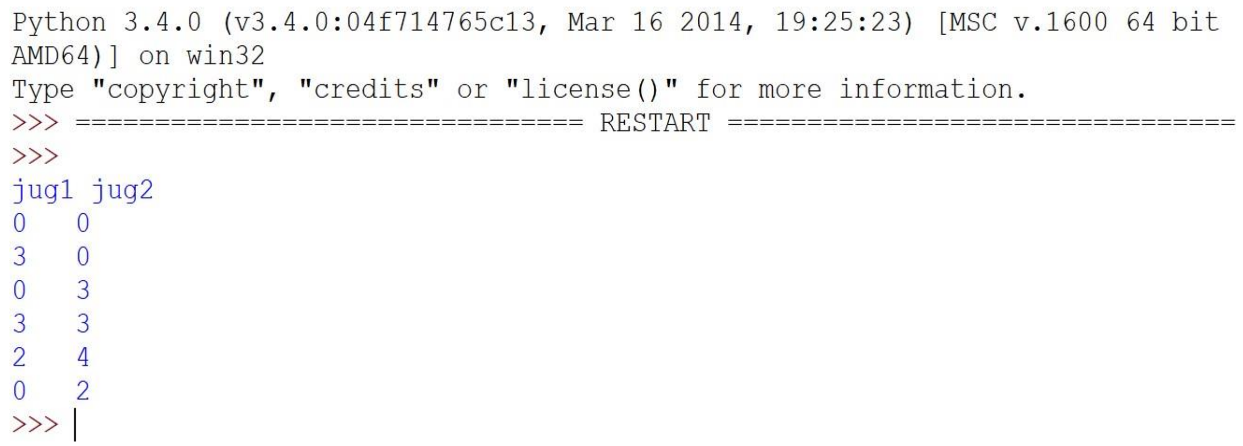
AIM:

Write a program to solve water jug problem.

PYTHON CODE:

```
def pour(jug1,jug2):
max1,max2,fill=3,4,2
print(jug1," ",jug2)     if
jug2 is fill:
        return     elif jug2 is
max2:         pour(0,jug1)
elif jug1!=0 and jug2 is 0:
        pour(0,jug1)     elif
jug1 is fill:
pour(jug1,0)     elif
jug1<max1:
pour(max1,jug2)     elif
jug1<(max2-jug2):
pour(0,(jug1+jug2))
else:
        pour(jug1-(max2-jug2),(max2-jug2)+jug2)
print("jug1 jug2") pour(0,0)
```

OUTPUT:



```
Python 3.4.0 (v3.4.0:04f714765c13, Mar 16 2014, 19:25:23) [MSC v.1600 64 bit
AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
jug1 jug2
0   0
3   0
0   3
3   3
2   4
0   2
>>> |
```

PRACTICAL NO.- 6

Write a program to shuffle deck of cards.

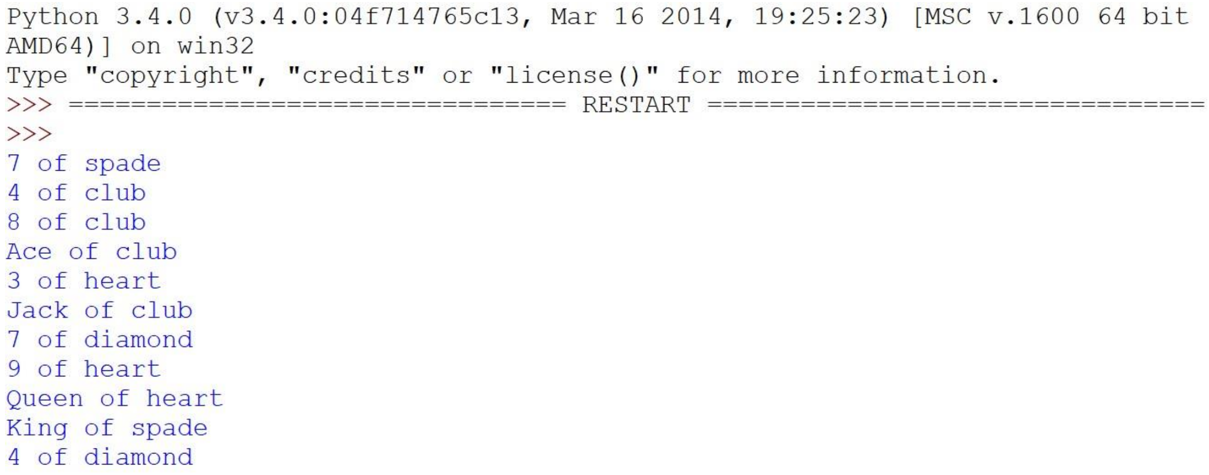
AIM:

Write a program to shuffle deck of cards.

PYTHON CODE:

```
import random cardfaces=[]
suits=["heart","diamond","club","spade"]
royals=["Jack","Queen","King","Ace"]
deck=[]
for i in range(2,11): #2 to 10
cardfaces.append(str(i)) for j
in range(4):
    cardfaces.append(royals[j]) for k in
range(4):      for l in range(13):
card=cardfaces[l]+" of "+suits[k]
deck.append(card)
random.shuffle(deck) for m in
range(52):      print(deck[m])
```

OUTPUT:



```
Python 3.4.0 (v3.4.0:04f714765c13, Mar 16 2014, 19:25:23) [MSC v.1600 64 bit
AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>> 
7 of spade
4 of club
8 of club
Ace of club
3 of heart
Jack of club
7 of diamond
9 of heart
Queen of heart
King of spade
4 of diamond
```

PRACTICAL NO.-7

AIM:-

Derive the expressions based on Associative law

The Associative Law: (“The parentheses shift and the numbers do not ").

We have learned to add two numbers at a time, but when we have three or more numbers to add, where do we begin? Does it matter? This is why the Associative Law was created.

For example, consider 3 + 10 + 2

You could first add 3 and 10 to get 13. Then add the result to 2 and obtain 15.

(3 + 10) + 2 = (13) + 2 = 15

Or you could first add 10 and 2 to get 12. Then add the result to 3 to get 15.

3 + (10 + 2) = 3 + (12) = 15

In both cases, we obtained the same answer.

(3 + 10) + 2 = 3 + (10 + 2)

Notice that the numbers: 3, 10, and 2 did not move.

What DID move was the parentheses.

In the first case, the parentheses were associated with the first two numbers 3 and 10.

The second time we tried the problem, they were placed around (associated with) the 10 and 2.

The Associative Law allows you to move parentheses as long as the numbers do not move.

For Example:-

Associative Law is similar to someone moving among a group of people associating with two different people at a time. You talk to Will and Nit for a while, then move your attention to Nit and Mith. Mith is next to Brij, so you chat with the two of them. The people remain standing in the same place as you turn your head to converse with different people. You associate your attention to two people at a time, but can move your attention and not move the people.

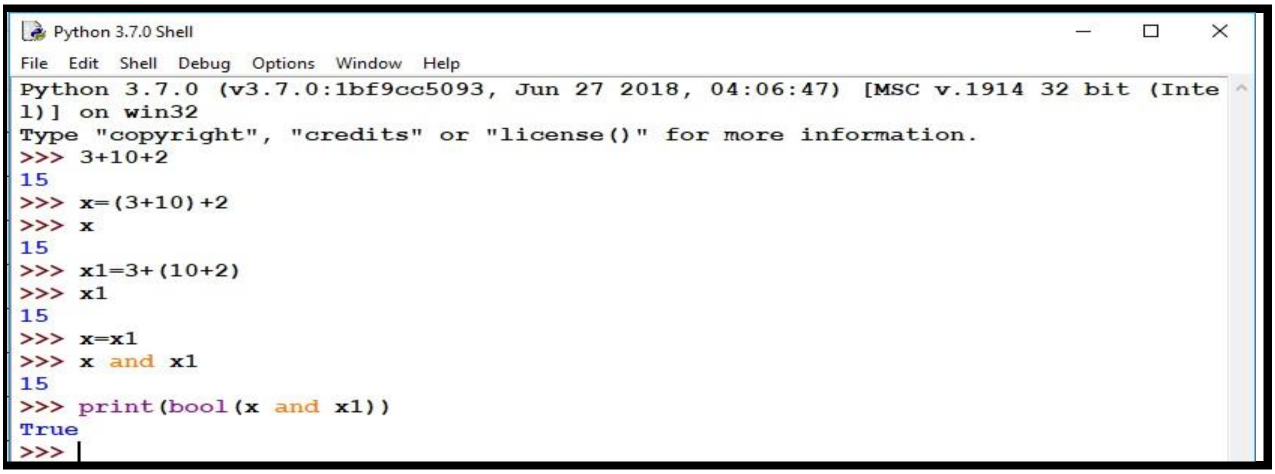
<u>Python Source Code:</u>

Examples of the Associative Law of Addition:

Look at both sides of the equation in the first step.

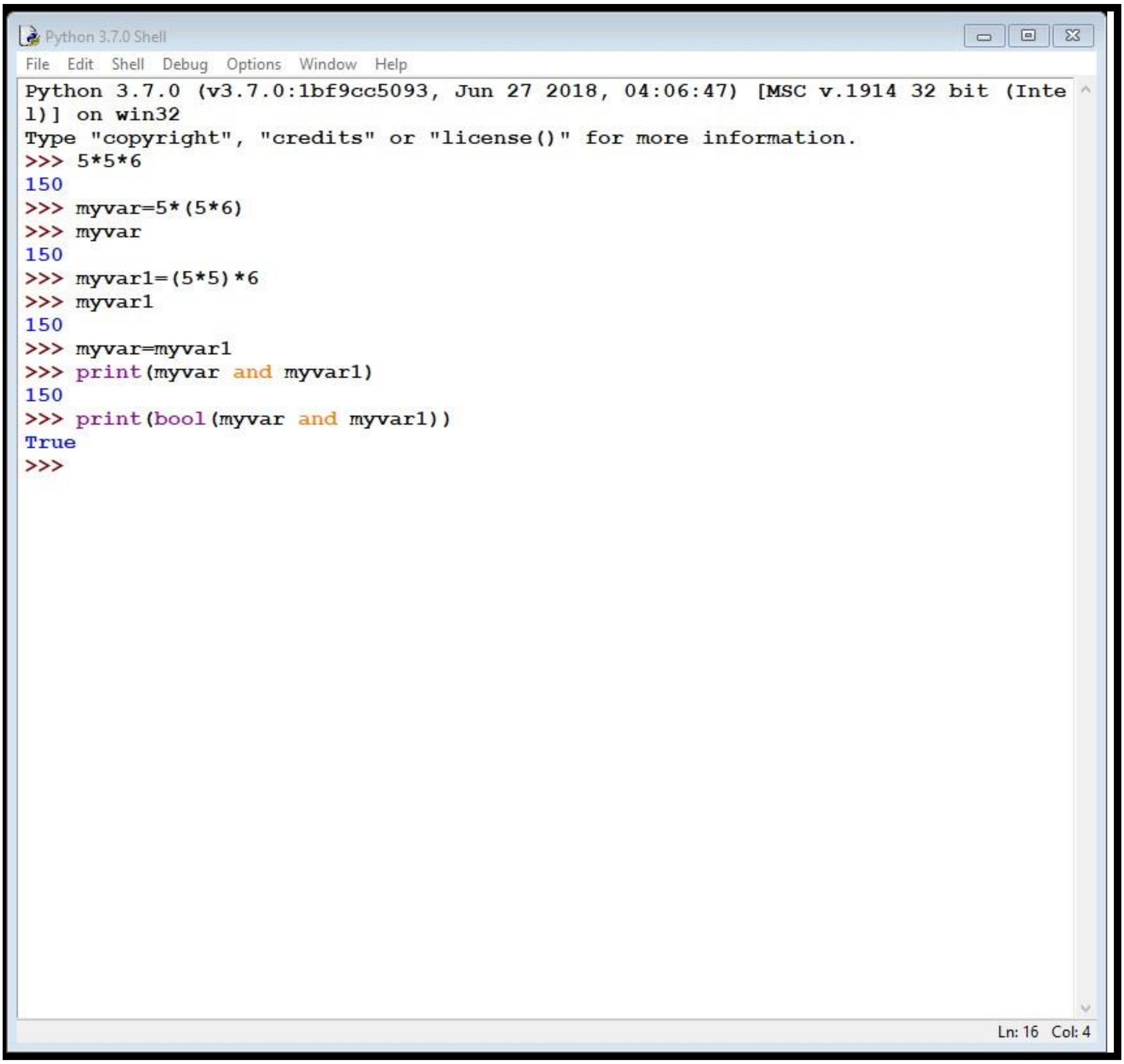
Parentheses move, but the numbers stay in the same order.

3+10+2=15



```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 3+10+2
15
>>> x=(3+10)+2
>>> x
15
>>> x1=3+(10+2)
>>> x1
15
>>> x=x1
>>> x and x1
15
>>> print(bool(x and x1))
True
>>> 
```

Examples of the Associative Law of Multiplication:
5*5*6=150



```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 5*5*6
150
>>> myvar=5*(5*6)
>>> myvar
150
>>> myvar1=(5*5)*6
>>> myvar1
150
>>> myvar=myvar1
>>> print(myvar and myvar1)
150
>>> print(bool(myvar and myvar1))
True
>>>
```

<u>KEY IDEA:</u>

In the Associative Law, the parentheses move but the numbers or letters do not. The Associative Law works when we add or multiply. It does NOT work when we subtract or divide.

PRACTICAL NO.-8

AIM:-

Derive the expressions based on Distributive law

The Distributive Law:

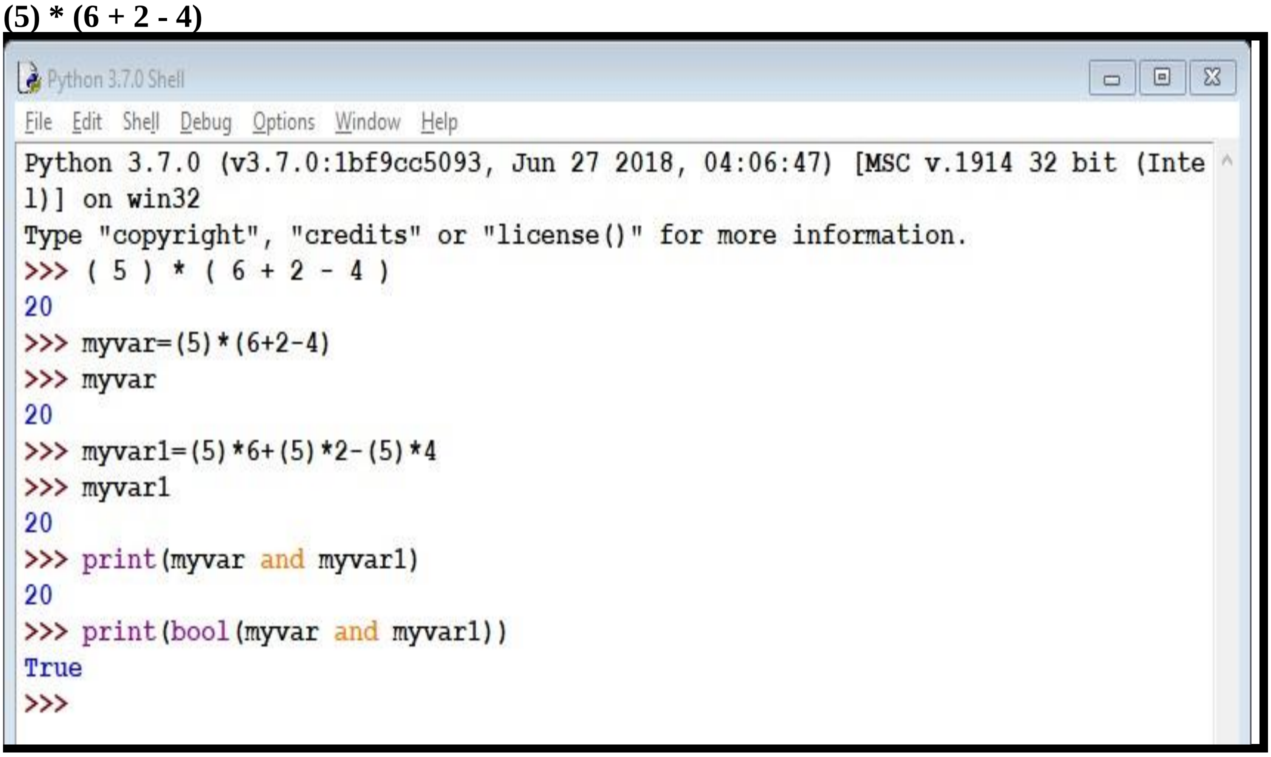
("Multiply everything inside parentheses by what is outside it")

Think of a delivery truck. It must move from the warehouse to several distributors along its route unloading its merchandise at each business. When the truck has unloaded at one stop, it moves to the next stop, unloads, and moves on until all locations have been visited. The distributive law is somewhat like a delivery truck, it is distributing multiplication among terms. The truck is outside the parentheses and the businesses are inside separated by plus and minus signs. When we multiply two numbers, each of the numbers is called a factor.

When (5) and (2) are multiplied producing 10, the 5 is one factor and the 2 is another factor. Now when we multiply (5) * (6 + 2 - 4) the 5 is one factor, but the other factor is an addition and subtraction problem: 6 + 2 - 4. The 6, 2, and 4 are not factors. They are joined together with addition signs and a subtraction sign making them "terms".

Example:-

(5) * (6 + 2 - 4)



```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ( 5 ) * ( 6 + 2 - 4 )
20
>>> myvar=(5)*(6+2-4)
>>> myvar
20
>>> myvar1=(5)*6+(5)*2-(5)*4
>>> myvar1
20
>>> print(myvar and myvar1)
20
>>> print(bool(myvar and myvar1))
True
>>>
```

This is the idea of the distributive law. When you have parentheses in which there is addition and/or subtraction, and when there is a factor outside of the parentheses, the factor may be distributed to all terms inside the parentheses. Remember "terms" are separated by addition or subtraction signs. In short, multiply every term inside the parentheses by the factor outside it.

KEY IDEA

The distributive law involves a number or variable outside of parentheses (a factor) and numbers or variables inside parentheses separated by addition and/or subtraction signs (terms).

Multiply every term inside the parentheses by the factor outside it. Thanks to the distributive 5(2 + 6) will produce the same result as 5(2) + 5(6).