1. **Write a program to implement DFS and BFS.**

```
# Program to print BFS traversal

# from a given source vertex. BFS(int s)

# traverses vertices reachable from s.

from collections import defaultdict

# This class represents a directed graph

# using adjacency list representation

class Graph:

 # Constructor

 def __init__(self):

 # default dictionary to store graph

 self.graph = defaultdict(list)

 # function to add an edge to graph

 def addEdge(self,u,v):

 self.graph[u].append(v)

 # Function to print a BFS of graph

 def BFS(self, s):

 # Mark all the vertices as not visited

 visited = [False] * (max(self.graph) + 1)

 # Create a queue for BFS

 queue = []

 # Mark the source node as
 # visited and enqueue it
 queue.append(s)
 visited[s] = True
 while queue:
 # Dequeue a vertex from
 # queue and print it
s = queue.pop(0)
 print (s, end = " ")
 # Get all adjacent vertices of the
 # dequeued vertex s. If a adjacent
 # has not been visited, then mark it
 # visited and enqueue it
 for i in self.graph[s]:
 if visited[i] == False:
 queue.append(i)
 visited[i] = True
# Driver code
# Create a graph given in
# the above diagram
g = Graph()
g.addEdge(0, 1)
g.addEdge(0, 2)
g.addEdge(1, 2)
```

```
g.addEdge(2, 0)

g.addEdge(2, 3)

g.addEdge(3, 3)

print ("Following is Breadth First Traversal"

 " (starting from vertex 2)")

g.BFS(2)
```

Output:

```
Following is Breadth First Traversal (starting from vertex 2)

> 3

2 0 3 1 3

>
```

2. Write a Program to find the solution for travelling salesman Problem

```
# program to implement traveling salesman

# problem using naive approach.

from sys import maxsize

from itertools import permutations

V = 4

# implementation of traveling Salesman Problem

def travellingSalesmanProblem(graph, s):

 # store all vertex apart from source vertex

 vertex = []

 for i in range(V):

 if i != s:
```

```
 vertex.append(i)
 # store minimum weight
 min_path = maxsize
 next_permutation=permutations(vertex)
 for i in next_permutation:
 # store current Path weight(cost)
 current_pathweight = 0
# compute current path weight
 k = s
 for j in i:
 current_pathweight += graph[k][j]
 k = j
 current_pathweight += graph[k][s]
 # update minimum
 min_path = min(min_path, current_pathweight)

 return min_path
# Driver Code
if __name__ == "__main__":
 # matrix representation of graph
 graph = [[0, 10, 15, 20], [10, 0, 35, 25],
 [15, 35, 0, 30], [20, 25, 30, 0]]
 s = 0
 print(travellingSalesmanProblem(graph, s))
```

Output

```
80
```

3. Write a program to find the solution for wampus world problem

Not added yet

4. Write a program to implement 8 puzzle problem

```
class Solution:
def solve(self, board):
dict = {}
flatten = []
for i in range(len(board)):
flatten += board[i]
flatten = tuple(flatten)
dict[flatten] = 0
if flatten == (0, 1, 2, 3, 4, 5, 6, 7, 8):
return 0
return self.get_paths(dict)
def get_paths(self, dict):
cnt = 0
while True:
current_nodes = [x for x in dict if dict[x] == cnt]
if len(current_nodes) == 0:
```

```
return -1
for node in current_nodes:
next_moves = self.find_next(node)
for move in next_moves:
if move not in dict:
dict[move] = cnt + 1
if move == (0, 1, 2, 3, 4, 5, 6, 7, 8):
return cnt + 1
cnt += 1
def find_next(self, node):
moves = {
0: [1, 3],
1: [0, 2, 4],
2: [1, 5],
3: [0, 4, 6],
4: [1, 3, 5, 7],
5: [2, 4, 8],
6: [3, 7],
7: [4, 6, 8],
8: [5, 7],
}
results = []
pos_0 = node.index(0)
```

```
for move in moves[pos_0]:
new_node = list(node)
new_node[move], new_node[pos_0] = new_node[pos_0], new_node[move]
results.append(tuple(new_node))
return results
ob = Solution()
matrix = [
[3, 1, 2],
[4, 7, 5],
[6, 8, 0]
]
print(ob.solve(matrix))
```

Input:

```
matrix = [
[3, 1, 2],
[4, 7, 5],
[6, 8, 0] ]
```

Output:

```
4
```

5. Write a program to implement Towers of Hanoi problem

```
# Recursive Python function to solve tower of hanoi
def TowerOfHanoi(n , from_rod, to_rod, aux_rod):
 if n == 1:
```

```
 print("Move disk 1 from rod",from_rod,"to rod",to_rod)
 return
 TowerOfHanoi(n-1, from_rod, aux_rod, to_rod)
 print("Move disk",n,"from rod",from_rod,"to rod",to_rod)
 TowerOfHanoi(n-1, aux_rod, to_rod, from_rod)


# Driver code
n = 4
TowerOfHanoi(n, 'A', 'C', 'B')
# A, C, B are the name of rods
```

Output

```
Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C
Move disk 3 from rod A to rod B
Move disk 1 from rod C to rod A
Move disk 2 from rod C to rod B
Move disk 1 from rod A to rod B
Move disk 4 from rod A to rod C
Move disk 1 from rod B to rod C
Move disk 2 from rod B to rod A
Move disk 1 from rod C to rod A
Move disk 3 from rod B to rod C
Move disk 1 from rod A to rod B

Move disk 2 from rod A to rod C

Move disk 1 from rod B to rod C
```

Output:

```
Tower of Hanoi Solution for 4 disks:

A: [4, 3, 2, 1] B: [] C: []

Move disk from rod A to rod B

A: [4, 3, 2] B: [1] C: []

Move disk from rod A to rod C

A: [4, 3] B: [1] C: [2]

Move disk from rod B to rod C

A: [4, 3] B: [] C: [2, 1]

Move disk from rod A to rod B

A: [4] B: [3] C: [2, 1]

Move disk from rod C to rod A

A: [4, 1] B: [3] C: [2]

Move disk from rod C to rod B

A: [4, 1] B: [3, 2] C: []

Move disk from rod A to rod B

A: [4] B: [3, 2, 1] C: []

Move disk from rod A to rod C

A: [] B: [3, 2, 1] C: [4]

Move disk from rod B to rod C
```

```
A: [] B: [3, 2] C: [4, 1]

Move disk from rod B to rod A

A: [2] B: [3] C: [4, 1]

Move disk from rod C to rod A

A: [2, 1] B: [3] C: [4]

Move disk from rod B to rod C

A: [2, 1] B: [] C: [4, 3]

Move disk from rod A to rod B

A: [2] B: [1] C: [4, 3]

Move disk from rod A to rod C

A: [] B: [1] C: [4, 3, 2]

Move disk from rod B to rod C

A: [] B: [] C: [4, 3, 2, 1]
```

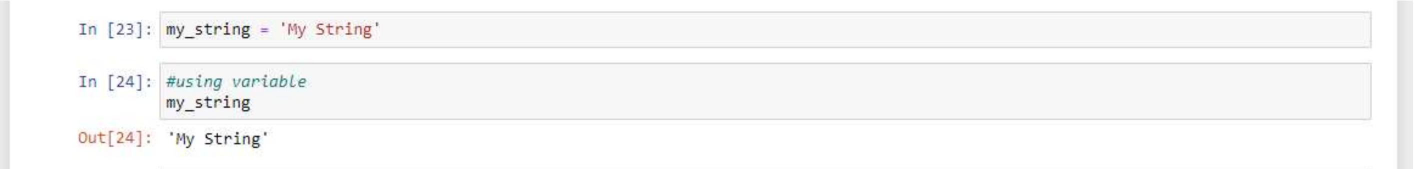
6. Define a string and assign it to a variable, e.g. my_string = ‘My String’ (but put something more interesting in the string). Print the contents of this variable in two ways:

(a) first by simply typing the variable name and pressing enter, then

Code:

```
#declare a variable

my_string = 'My String'

#using variable

my_string
```

Output:



```
In [23]: my_string = 'My String'

In [24]: #using variable
         my_string

Out[24]: 'My String'
```

(b) by using the print statement.

Code:

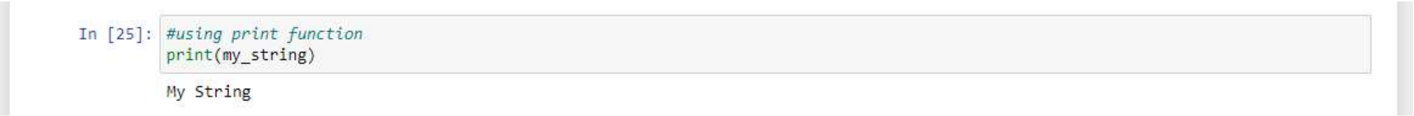
#declare a variable

my_string = 'My String'

#using print function

print(my_string)

Output:



```
In [25]: #using print function
         print(my_string)

         My String
```

7. Define set to be the list of words [‘she’, ‘sells’, ‘sea’, ‘shells’, ‘by’, ‘the’, ‘sea’, ‘shore’]. Now, write code to perform the following tasks:

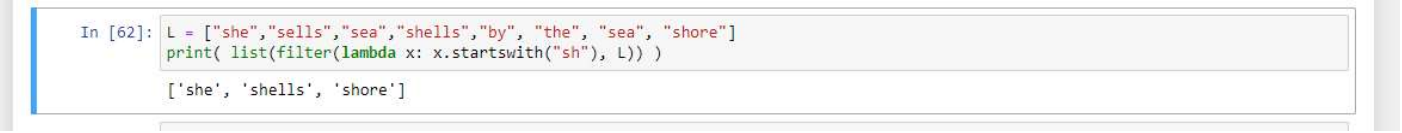
(a) Print all words beginning with ‘sh’.

Code:

L = ["she","sells","sea","shells","by", "the", "sea", "shore"]

print(list(filter(lambda x: x.startswith("sh"), L)))

Output:



```
In [62]: L = ["she","sells","sea","shells","by", "the", "sea", "shore"]
         print( list(filter(lambda x: x.startswith("sh"), L)) )

         ['she', 'shells', 'shore']
```

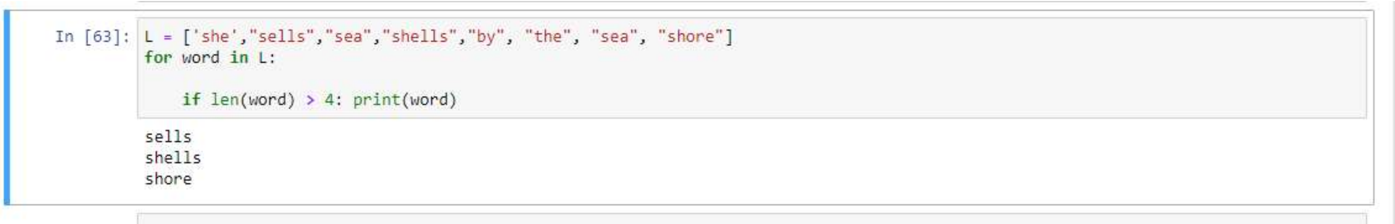
(b) Print all words longer than four characters.

Code:

L = ['she',"sells","sea","shells","by", "the", "sea", "shore"]

```
for word in L:

        if len(word) > 4: print(word)
```

Output:



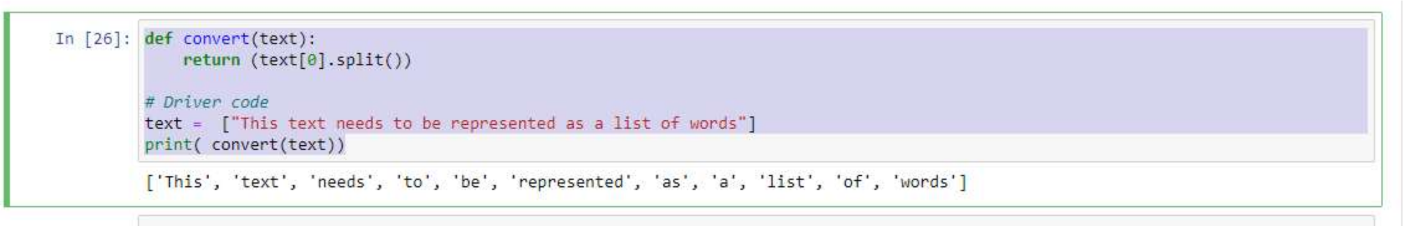
```
In [63]: L = ['she',"sells","sea","shells","by", "the", "sea", "shore"]
         for word in L:

             if len(word) > 4: print(word)

         sells
         shells
         shore
```

8. Program to represent text as list of words.

Code:

```
def convert(text):

    return (text[0].split())

text =  ["This text needs to be represented as a list of words"]

print( convert(text))
```

Output:



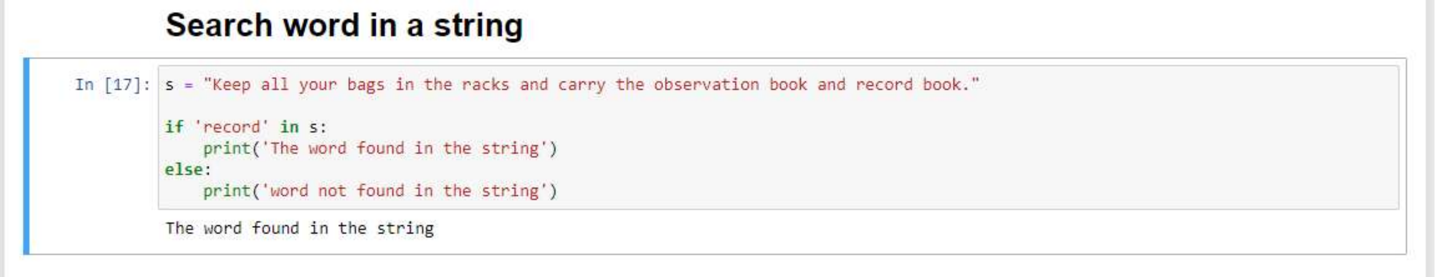
```
In [26]: def convert(text):
             return (text[0].split())

         # Driver code
         text =  ["This text needs to be represented as a list of words"]
         print( convert(text))

         ['This', 'text', 'needs', 'to', 'be', 'represented', 'as', 'a', 'list', 'of', 'words']
```

9. Program to search text.

Code:

```
s = "Keep all your bags in the racks and carry the observation book and record book."

if 'record' in s:

    print('The word found in the string')

else:
    print('word not found in the string')
```

Output:



Search word in a string

```
In [17]: s = "Keep all your bags in the racks and carry the observation book and record book."

         if 'record' in s:
             print('The word found in the string')
         else:
             print('word not found in the string')

         The word found in the string
```

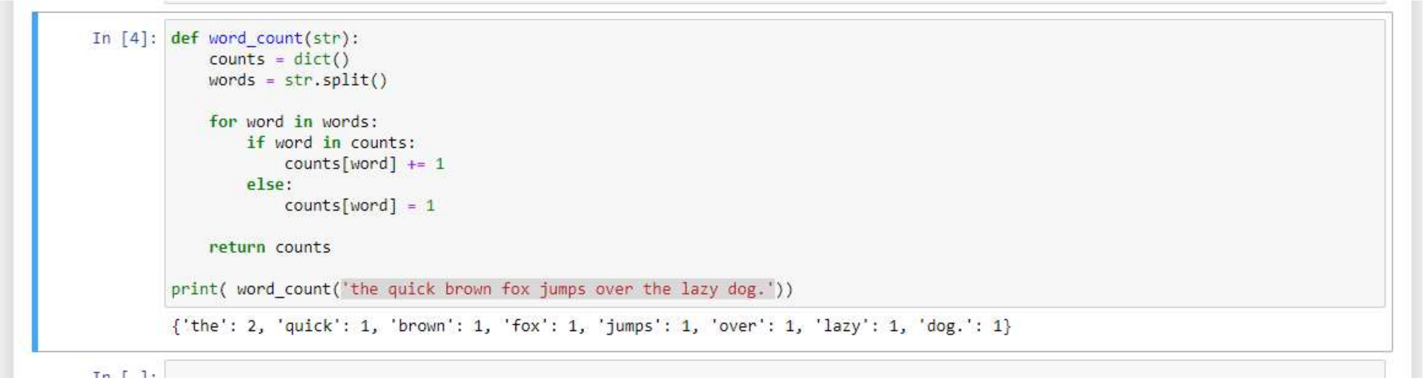
10. Program to count vocabulary and sorting vocabulary.

(a) Counting unsorted words in a string

Code:

```
def word_count(str):
    counts = dict()
    words = str.split()
    for word in words:
        if word in counts:
            counts[word] += 1
        else:
            counts[word] = 1
    return counts
print( word_count('the quick brown fox jumps over the lazy dog.'))
```

Output:



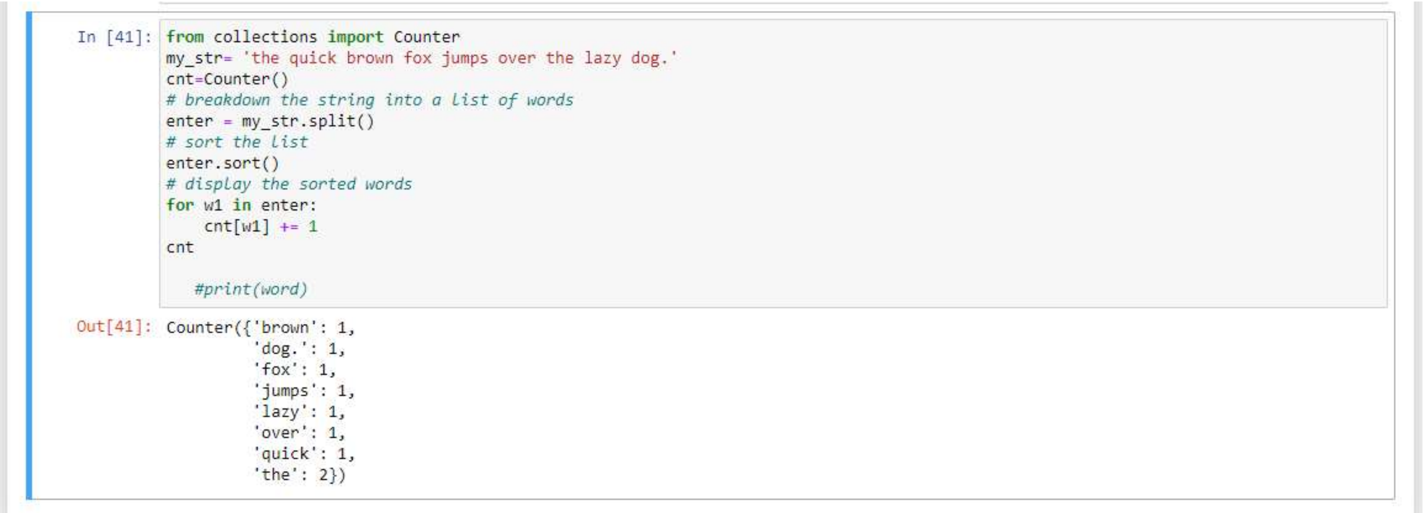
```
In [4]: def word_count(str):
            counts = dict()
            words = str.split()

            for word in words:
                if word in counts:
                    counts[word] += 1
                else:
                    counts[word] = 1

            return counts

        print( word_count('the quick brown fox jumps over the lazy dog.'))

        {'the': 2, 'quick': 1, 'brown': 1, 'fox': 1, 'jumps': 1, 'over': 1, 'lazy': 1, 'dog.': 1}
```

(b) Counting sorted words in a string

Code:

```
from collections import Counter

my_str= 'the quick brown fox jumps over the lazy dog.'

cnt=Counter()

# breakdown the string into a list of words

enter = my_str.split()

# sort the list

enter.sort()

# display the sorted words

for w1 in enter:

   cnt[w1] += 1

cnt

       #print(word)
```

Output:



```
In [41]: from collections import Counter
         my_str= 'the quick brown fox jumps over the lazy dog.'
         cnt=Counter()
         # breakdown the string into a list of words
         enter = my_str.split()
         # sort the list
         enter.sort()
         # display the sorted words
         for w1 in enter:
             cnt[w1] += 1
         cnt

            #print(word)
```

```
Out[41]: Counter({'brown': 1,
                  'dog.': 1,
                  'fox': 1,
                  'jumps': 1,
                  'lazy': 1,
                  'over': 1,
                  'quick': 1,
                  'the': 2})
```