In [ ]:
```
permutation
```

In [9]:
```
def permutation(lst, r):
    if r==0:
        return [[]]
    if len(lst)== 0:
        return []
    l=[]
    for i in range(len(lst)):
        m=lst[i]
        remLst=lst[:i]+lst[i+1:]
        for p in permutation(remLst, r - 1):
            l.append([m] + p)
    return l
data=[1, 2, 3]
r=2
p=permutation(data, r)
for i in range(0, len(p)):
    print(p[i])
```

```
[1, 2]
[1, 3]
[2, 1]
[2, 3]
[3, 1]
[3, 2]
```

In [12]:
```
def permute(elem,path=[]):
    if not elem:
        print(path)
    else:
        for i in range(len(elem)):
            permute(elem[:i]+elem[i+1:], path+[elem[i]])
def permute_str(s):
    permute(list(s))
def permute_n(nums):
    permute(nums)
permute_string("ABC")
permute_n([1,2,3])
```

```
['A', 'B', 'C']
['A', 'C', 'B']
['B', 'A', 'C']
['B', 'C', 'A']
['C', 'A', 'B']
['C', 'B', 'A']
[1, 2, 3]
[1, 3, 2]
[2, 1, 3]
[2, 3, 1]
[3, 1, 2]
[3, 2, 1]
```

In [ ]:
```
combimnations
```

```
In [14]: def comb(arr, k):
             if k==0:
                 return [[]]
             if len(arr)<k:
                 return []
             if len(arr)==k:
                 return [arr]
             result=[]
             for i in range(len(arr)):
                 f=arr[i]
                 rem=arr[i+1:]
                 for c in comb(rem,k-1):
                     result.append([f] + c)
             return result
         m=input("Enter elements of separated by spaces: ").split()
         k=int(input("Enter the len of comb: "))
         combination= comb(m, k)
         print(combination)
```

```
Enter elements of separated by spaces: a b c
Enter the len of comb: 2
[['a', 'b'], ['a', 'c'], ['b', 'c']]
```

```
In [ ]: subset generator
```

```
In [18]: def Subsett(A, res, subset, index):
             res.append(subset[:])
             for i in range(index, len(A)):
                 subset.append(A[i])
                 Subsett(A,res,subset,i+1)
                 subset.pop()
         def subsets(A):
             subset=[]
             res=[]
             index=0
             Subsett(A, res, subset, index)
             return res
         array=[1,2,3]
         res=subsets(array)
         for subset in res:
             print(*subset)
```

```

1
1 2
1 2 3
1 3
2
2 3
3
```

```
In [ ]: sudoku solver
```

In [19]:
```
def is_valid(board, row, col, num):
    for x in range(9):
        if board[row][x]==num or board[x][col]==num:
            return False
    start_row, start_col=3 * (row//3), 3 * (col//3)
    for i in range(3):
        for j in range(3):
            if board[start_row + i][start_col + j]==num:
                return False
    return True
def solve_sudoku(board):
    for row in range(9):
        for col in range(9):
            if board[row][col]== 0:
                for num in range(1, 10):
                    if is_valid(board, row, col, num):
                        board[row][col]=num
                        if solve_sudoku(board):
                            return True
                        board[row][col]=0
                return False
    return True
def print_board(board):
    for row in board:
        print(" ".join(str(num) for num in row))
sudoku_board = [
    [5, 3, 0, 0, 7, 0, 0, 0, 0],
    [6, 0, 0, 1, 9, 5, 0, 0, 0],
    [0, 9, 8, 0, 0, 0, 0, 6, 0],
    [8, 0, 0, 0, 6, 0, 0, 0, 3],
    [4, 0, 0, 8, 0, 3, 0, 0, 1],
    [7, 0, 0, 0, 2, 0, 0, 0, 6],
    [0, 6, 0, 0, 0, 0, 2, 8, 0],
    [0, 0, 0, 4, 1, 9, 0, 0, 5],
    [0, 0, 0, 0, 8, 0, 0, 7, 9]
]

if solve_sudoku(sudoku_board):
    print_board(sudoku_board)
else:
    print("No solution exists")
```

```
5 3 4 6 7 8 9 1 2
6 7 2 1 9 5 3 4 8
1 9 8 3 4 2 5 6 7
8 5 9 7 6 1 4 2 3
4 2 6 8 5 3 7 9 1
7 1 3 9 2 4 8 5 6
9 6 1 5 3 7 2 8 4
2 8 7 4 1 9 6 3 5
3 4 5 2 8 6 1 7 9
```

In [ ]: