Palindrome Partitioning

Hint: Try for every possible substring and check if it is palindrome add it to the list and then move forward.

Watch this if you have trouble understanding

C++

```
 class Solution {
public:
```

```
vector<vector<string>> res ;
    //function for checking if string is pallindrome
    bool isPallindrome(string s1)
       {
             int i = 0, j = s1.length()-1;
             while(i <= j)
             {
                    char a = s1[i];
                    char b = s1[j];
                    if(a != b)
                    return false;
            i++; j--;
          }
             return true;
       }
       void makePartition(string s, int i, vector<string> l1)
       {
             int j;

             if(i >= s.length())
             {
             res.push_back(l1);
                    return;
             }
             for(j = i; j< s.length(); j++)
             {

           //checking if substring is pallindrome then yes we can partition it
                    if(isPallindrome(s.substr(i ,j- i + 1)))
                    {
                          l1.push_back(s.substr(i, j- i +1));
                          makePartition(s , j + 1, l1);
                          l1.pop_back();
                    }
             }
       }
  vector<vector<string>> partition(string s) {
        vector<string> l1;
        makePartition(s , 0 , l1);
        return res;
    }
};
```

Java

```
class Solution {
       List<List<String>> res =  new ArrayList<List<String>>();
       public boolean isPallindrome(String s1)
       {
             int i = 0, j = s1.length()-1;
             while(i <= j)
```

```
            {
                    char a = s1.charAt(i);
                    char b = s1.charAt(j);
                    if(a != b)
                    return false;
           i++; j--;
         }
            return true;
        }
        public void makePartition(String s, int i, List<String> l1)
        {
            int j;

            if(i >= s.length())
            {
            res.add(new ArrayList<String>(l1));
                    return;
            }
            for(j = i; j< s.length(); j++)
            {
           //checking if substring is pallindrom the yes we can partition
                    if(isPallindrome(s.substring(i , j+1)))
                    {
                            l1.add(s.substring(i, j+1));
                            makePartition(s , j + 1, l1);
                            l1.remove(l1.size() - 1);
                    }
            }
        }
    public List<List<String>> partition(String s) {
        List<String> l1 = new ArrayList<String>();
        makePartition(s , 0 , l1);
        return res;
    }
}
```

Python

```
class Solution:
    def partition(self, s: str) -> List[List[str]]:
        def isPalindrome(string):
            start = 0;
            end = len(string)-1
            while(start < end):
                if(string[start] != string[end]):
                    return False
                start += 1
                end -= 1
            return True

        def makePartitions(curr, s):

            if(len(s) == 0):
                result.append(curr[:])
                return

            for i in range(1,len(s)+1):
                if(isPalindrome(s[:i])):
                    makePartitions(curr + [s[:i]], s[i:])

        result = []
        makePartitions([], s)
        return result
```

Generate Parentheses

Hint: We should observe that for a string to be balanced never ever the right parentheses(‘)’) should be greater than the left one (‘(‘).
Now try to make the combinations by using this fact.

Watch this if you have trouble understanding

C++

```
class Solution {
 public:
    vector<string> res ;
    void generate(string s1, int l, int r , int n)
    {

        if(s1.length() >= 2 * n)
        {
            res.push_back(s1);
            return;
        }
        //checking and picking left brackets
        if(l < n)
        {

            generate(s1 + "(", l + 1, r , n);
        }
        //checking that right brackets are always less that left one
        if(l>r)
        {

            generate(s1 + ")", l , r+1 , n);
        }
    }
     vector<string> generateParenthesis(int n) {
        generate("" , 0, 0 , n);
        return res;
    }
};
```

Java

```
class Solution {
    List<String> res = new ArrayList<String>();
    public void generate(String s1, int l, int r , int n)
    {

        if(s1.length() >= 2 * n)
        {
            res.add(new String(s1));
            return;
        }
        //checking and picking left elements
        if(l < n)
        {

            generate(s1 + "(", l + 1, r , n);
        }
        //checking left braces always greater right
        if(l > r)
        {

            generate(s1 + ")", l , r+1 , n);
        }
    }
    public List<String> generateParenthesis(int n) {
        generate("" , 0, 0 , n);
        return res;
    }
}
```

Python

```
class Solution(object):
    def generateParenthesis(self, n):
        res = []
        def generate(s1 , l, r , n):
            if len(s1) == 2*n:
```

```
        res.append(s1[:])
        return
    if l < n:
        generate(s1 + '(' , l+1 , r , n)
    if l > r:
        generate(s1 + ')', l, r+1 , n)
generate('' , 0 , 0 , n)
return res
```

Gray Code

Let’s see how we can make grey code for n = 2
For n = 1
we have only ‘0’, ‘1’
we will add 0 to n = 1 (greys code)
00 01
now we will add 1 to n = 1 (greys code) but from reverse
11 , 10
Similarly we will do for anyother n .
The base case would be n = 1 it means we will use n = 1 to make n = 2 and so on.

C++

```
class Solution {
 public:
    vector<string> get(int n)
    {
        if(n == 1)
        {
            vector<string> l2;
            l2.push_back("0");
            l2.push_back("1");
            return l2;
        }
        vector<string> l3;
        vector<string> l4;

        l3 = get(n-1);

        int size = l3.size();

        int i;
        //adding 0 to n -1 th grey code
        for(i = 0; i < size; i++)
        {
            string temp = l3[i];
            l4.push_back("0" + temp);
        }
        //adding 1
        for(i = size-1; i >= 0; i--){
            string temp = l3[i];
            l4.push_back("1" + temp);
        }

        return l4;
    }
    vector<int> grayCode(int n) {
        vector<string> l1;
        l1 = get(n);
        vector<int> res ;
        int i;
        //converting binary string to integer
        for(i = 0; i < l1.size(); i++)
        {
            int x = stoi(l1[i],0,2);
            res.push_back(x);
        }
        return res;
    }
};
```

Java

```
class Solution {
    public List<String> get(int n)
    {
        if(n == 1)
        {
            List<String> l2 = new ArrayList<String>();
            l2.add("0");
            l2.add("1");
            return l2;
        }
        List<String> l3 = new ArrayList<String>();
        List<String> l4 = new ArrayList<String>();

        l3 = get(n-1);

        int size = l3.size();

        int i;
        //adding 0
        for(i = 0; i < size; i++)
        {
            String temp = l3.get(i);
            l4.add("0" + temp);
        }
        //adding 1
        for(i = size-1; i >= 0; i--){
            String temp = l3.get(i);
            l4.add("1" + temp);
        }

        return l4;
    }
    public List<Integer> grayCode(int n) {
        List<String> l1 = new ArrayList<String>();
        l1 = get(n);
        List<Integer> res = new ArrayList<Integer>();
        int i;
        for(i = 0; i < l1.size(); i++)
        {
            String temp = l1.get(i);
            res.add(Integer.parseInt(temp , 2));
        }
        return res;
    }
}
```

Python

```
class Solution(object):
    def grayCode(self, n):
        def get(n):
            if n == 1:
               l1 = []
```

```
                   l1.append("0")
                   l1.append("1")
                   return l1
            l2 = get(n-1)
            l3 = []
            for i in range(len(l2)):
                 temp = l2[i]
                 l3.append("0" + temp)
            for i in range(len(l2)-1, -1 ,-1):
                   temp = l2[i]
                   l3.append("1" + temp)
            return l3
        l1 = get(n)
        res = []
        for i in range(len(l1)):
           res.append(int(l1[i], 2))
        return res
```

Sudko Solver

Hint: Go to every row and try filling every box with number from 1 to 9 and then check if it is possible to fill the box with this particular no.

C++

```
  class Solution {
public:
     void solveSudoku(vector<vector<char>> &board) {
        if(board.size() == 0)
            return;
        solve(board);
    }
     bool solve(vector<vector<char>> &board){
        for(int i = 0; i < board.size(); i++){
            for(int j = 0; j < board[0].size(); j++){
                if(board[i][j] == '.'){
                    for(char c = '1'; c <= '9'; c++){
                        if(isValid(board, i, j, c)){
                            board[i][j] = c;
                            if(solve(board))
                                return true;
                            else
                                board[i][j] = '.';
                        }
                    }
                    return false;
                }
            }
        }
        return true;
    }
     bool isValid(vector<vector<char>> &board, int row, int col, char c){
        for(int i = 0; i < 9; i++) {
            if(board[i][col] != '.' && board[i][col] == c)
                return false;
            if(board[row][i] != '.' && board[row][i] == c)
                return false;
            if(board[3 * (row / 3) + i / 3][ 3 * (col / 3) + i % 3] != '.' &&
board[3 * (row / 3) + i / 3][3 * (col / 3) + i % 3] == c)
                return false;
        }
        return true;
    }
};
```

Java

```
public class Solution {
    public void solveSudoku(char[][] board) {
        if(board == null || board.length == 0)
```

```
            return;
        solve(board);
    }
    public boolean solve(char[][] board){
        for(int i = 0; i < board.length; i++){
            for(int j = 0; j < board[0].length; j++){
                if(board[i][j] == '.'){
                    for(char c = '1'; c <= '9'; c++){
                        if(isValid(board, i, j, c)){
                            board[i][j] = c;
                            if(solve(board))
                                return true;
                            else
                                board[i][j] = '.';
                        }
                    }
                    return false;
                }
            }
        }
        return true;
    }
    private boolean isValid(char[][] board, int row, int col, char c){
        for(int i = 0; i < 9; i++) {
            if(board[i][col] != '.' && board[i][col] == c)
                return false;
            if(board[row][i] != '.' && board[row][i] == c)
                return false;
            if(board[3 * (row / 3) + i / 3][ 3 * (col / 3) + i % 3] != '.' &&
board[3 * (row / 3) + i / 3][3 * (col / 3) + i % 3] == c)
                return false;
        }
        return true;
    }
}
```

Python

```
class Solution(object):
    def solveSudoku(self, board):
        if not board or len(board) == 0:
            return
        self.solve(board)

    def solve(self, board):
        for i in range(len(board)):
            for j in range(len(board[0])):
                if board[i][j] == '.':
                    for c in "123456789":
                        if self.isValid(board, i, j, c):
                            board[i][j] = c
                            if self.solve(board):
                                return True
                            else:
                                board[i][j] = '.'
                    return False
        return True

    def isValid(self, board, x, y, c):
        for i in range(9):
            if board[i][y] == c:
                return False
        for j in range(9):
            if board[x][j] == c:
                return False
        for i in range(3):
            for j in range(3):
                if board[(x/3)*3 + i][(y/3)*3 + j] == c:
                    return False

        return True
```