import java.io.\*;

import java.math.\*;

import java.security.\*;

import java.text.\*;

import java.util.\*;

import java.util.concurrent.\*;

import java.util.regex.\*;

    // Write your code here

    // Java program to find palindromic substrings of a string

class Palindrome

{

    // Returns total number of palindrome substring of

    // length greater then equal to 2

    static int CountPS(char str[], int n)

    {

        // create empty 2-D matrix that counts all palindrome

        // substring. dp[i][j] stores counts of palindromic

        // substrings in st[i..j]

        int dp[][] = new int[n][n];

        // P[i][j] = true if substring str[i..j] is palindrome,

        // else false

        boolean P[][] = new boolean[n][n];

        // palindrome of single length

        for (int i= 0; i< n; i++)

            P[i][i] = true;

        // palindrome of length 2

        for (int i=0; i<n-1; i++)

        {

            if (str[i] == str[i+1])

            {

                P[i][i+1] = true;

                dp[i][i+1] = 1 ;

            }

        }

        // Palindromes of length more than 2. This loop is similar

        // to Matrix Chain Multiplication. We start with a gap of

        // length 2 and fill the DP table in a way that gap between

        // starting and ending indexes increases one by one by

        // outer loop.

        for (int gap=2 ; gap<n; gap++)

        {

            // Pick starting point for current gap

            for (int i=0; i<n-gap; i++)

            {

                // Set ending point

                int j = gap + i;

                // If current string is palindrome

                if (str[i] == str[j] && P[i+1][j-1] )

                    P[i][j] = true;

                // Add current palindrome substring ( + 1)

                // and rest palindrome substring (dp[i][j-1] + dp[i+1][j])

                // remove common palindrome substrings (- dp[i+1][j-1])

                if (P[i][j] == true)

                    dp[i][j] = dp[i][j-1] + dp[i+1][j] + 1 - dp[i+1][j-1];

                else

                    dp[i][j] = dp[i][j-1] + dp[i+1][j] - dp[i+1][j-1];

            }

        }

        // return total palindromic substrings

        return dp[0][n-1];

    }

    // Driver Method

    public static void main(String[] args)

    {

        String str = "abccba";

        System.out.println(CountPS(str.toCharArray(), str.length()));

    }

    }

public class Solution {

    public static void main(String[] args) throws IOException {

        BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

        BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

        String str = "abaab";

        int result = Palindrome.CountPS(str.toCharArray(), str.length());

        bufferedWriter.write(String.valueOf(result));

        bufferedWriter.newLine();

        bufferedReader.close();

        bufferedWriter.close();

    }

}