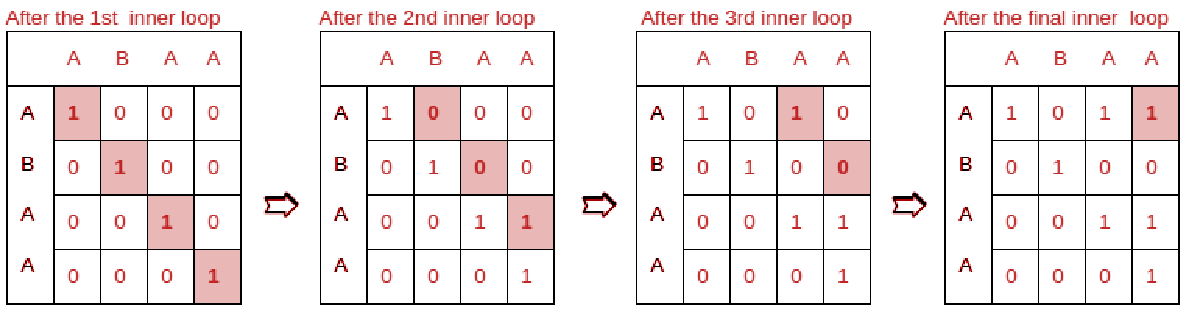
Dynamic Programming Approach

We will use the concept that if the substring STR[i..j] is a palindrome, then the substring STR[i + 1 .. j - 1] will be a palindrome as well. So we use this to formulate a dynamic programming solution where we expand around the center.

- If the string length is less than or equal to 1 then return the string, as one length string is always palindromic.
- Initialize a 'DP' array of data type boolean, DP[i][j] will store false if STR[i, j] is not palindromic, otherwise it will store true.
- Store all the diagonal elements (DP[i][i]) as true, as STR[i, i] will always be palindromic.
- For substring of length 2, check if STR[i] is equal to STR[i + 1] then store DP[i][i + 1] as true.
- Run a loop for the length of a substring greater than 2, fill the DP array diagonally.
- To calculate DP[i][j], check the value of DP[i + 1][j - 1], if the value is true and STR[i] is the same as STR[j], then we make DP[i][j] true, otherwise false.
- For every DP[i][j] true, update the length and starting index of the substring.
- Return the substring of the string having a starting index as start and of maximum length.

The DP Table filling for the string **ABAA** is shown below :



After the 1st inner loop

	A	B	A	A
A	**1**	0	0	0
B	0	**1**	0	0
A	0	0	**1**	0
A	0	0	0	**1**

After the 2nd inner loop

	A	B	A	A
A	1	**0**	0	0
B	0	1	**0**	0
A	0	0	1	**1**
A	0	0	0	1

After the 3rd inner loop

	A	B	A	A
A	1	0	**1**	0
B	0	1	0	**0**
A	0	0	1	1
A	0	0	0	1

After the final inner loop

	A	B	A	A
A	1	0	1	**1**
B	0	1	0	0
A	0	0	1	1
A	0	0	0	1

```
/*
    Function to find the longest palindromic substring of the given string
*/
function longestPalinSubstring(str)
    n = len(str)
    if n < 1
        return ""

    /*
```

```
          dp[i][j] will be true if str[i..j] is palindrome.
          Else dp[i][j] will be false.
*/
dp = [[False for j in range(n)] for i in range(n)]

maxLength = 1

//   Single letter is always palindromic.
for i in range(n)
          dp[i][i] = True

start = 0

//   Substring of length 2.
for i in range(n - 1)
          if str[i] == str[i + 1]
                    dp[i][i + 1] = True
                    if maxLength < 2
                              start = i
                              maxLength = 2

/*
          Check for lengths greater than 2.
          k is the length of the substring.
*/
for length in range(3,n + 1)

          //   Fix the starting index.
          for i in range(n - length +1)
                    //   Ending index of length 'length'.
                    j = i + length - 1

                    //   Condition of str[i, j] to be palindromic.
                    if (dp[i + 1][j - 1] is True and str[i] == str[j])
                        dp[i][j]= True

                        //   Update the starting index and the length.
                        if (length > maxLength)
                            start = i
                            maxLength = length

        return str[start,  start + maxLength]
```

Note: When there are multiple answers of the same length our algorithm will automatically return the first string because we are only updating the starting index when the new length is strictly greater than the current best. Hence, in case of ties our ‘start’ will remain unchanged.

Time Complexity : O(N ^ 2), where N is the length of the given string, as we are traversing the ‘DP’ array once.

Space Complexity : O(N ^ 2), where N is the length of the given string.We are using a ‘DP’ array of N*N dimensions.