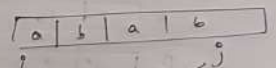
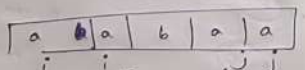
Brute force

ispalindrome (String s).

```
for (int i=0, j=s.length()-1;
     i<j; i++, j--){
  if (s[i] != s[j]){
    return false
  }
}
return true;
}
```



check equal or not if not return false



equal

if all equal and you come out of loop it is true

String longest palindrome (String s){

result string = bab l = 3

t = ~~ba~~ bab maxlen = 3

bab X not palindrom

Logic: iterate j for each i till j so we can take all substring and check if palindrome and if yes update result and max len.

output - bab

```
int maxlen = 0;
String result = "";
for (int i=0; i<s.length(); i++){
  String t = "";
  for (int j=i; j<s.length(); j++){
    t += s[j];
    int l = j-i+1;
    if (ispalindrome(t) && maxlen<l)
    {
      maxlen = l;
      result = "";
      result += t; } } return result; }
```

Better approach (with ~~recursion~~ recursion)

T.C. → $O(2^n)$

S.C. → $O(n)$ stack space.

for palindrome check using recursion
(so memoize this).

```
bool i,j){
  if(i>=j)
    return true;
  if(s[i]==s[j]){
    return solve(i+1,j-1);
  }
  else{return false;}
}
```

let us make substring.

~~string substring(int i, string s){~~

```
int dp[1001][1001];
```

(can be global or else RUNTIME ERROR)

```
solve(string s, int i, int j)
{
  if(i>=j){
  return true;}
  if(dp[i][j]!=-1){
    return dp[i][j];
  }
  if(s[i]==s[j]){
    return dp[i][j]=
  ispalindrome(s,i+1,j-1);
  }
  else{return dp[i][j]=false;
  }
}
```

D.P solution. t[i][j] → true → s[i...j] → Palindrome

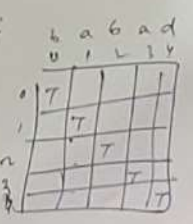
→ false → s[i...j] → Not palindrome

t[0][0] = true

t[1][1] = true

```
for(int i=0; i<n; i++){
  dp[i][i]=true;
}
```

```
for(int i=0; i<n; i++){
  for(int j=i; j<n; j++){
    if(s[i]==s[j] && dp[i+1][j-1]){
      dp[i][j]=true;
    }
```



Blueprint to use at maximum string question

```
for ( int L = 2 ; L <= s.length() ; L++ ) {
    for ( int i = 0 ; i < n - L + 1 ; i++ ) {
        int j = L + i - 1 ;
        if ( s[i] == s[j] && dp[i+1][j-1] || L == 2 ) {
            dp[i][j] = true ;
            maxLen = max ( maxLen , j - i + 1 ) ;
            start = i ;
        }
        else if ( s[i] == s[j] && dp[i+1][j-1] ) {
            dp[i][j] = true ;
            maxLen = max ( maxLen , j - i + 1 ) ;
            start = i ;
        }
        else {
            dp[i][j] = false ;
        }
    }
return sa.substr ( start , maxLen ) ;
}
```