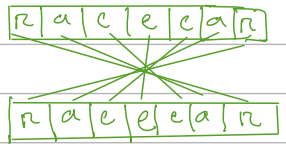
Longest Palindromic Substring:-

Input S:- "babad"

Output :- bab

A string is Palindrome if the string remains same even if we write it from the opposite direction.

For Example:- "race car"



It is a Palindrome.

We have to find the longest Palindrome.

Solution:-

The solution in this case is expanding window technique.

Time complexity $O(N^2)$.

Intuition:- The intuition is to expand around the centre. If we look at the Palindrome, we will see that the character's on the left and right of the center element has to be same. That's why we expand from the centre to keep track and check if we were to swap their positions would the string still be same or not.

Think of it like you are in the center of a parking lot and you look left once and then right to check if the colors of the cars are same or not. Would anyone know if you were to swap the positions of the car?

Code explaination:-

From the explaination above, we know, you have to look at your left and right. That means we have to cross reference with something. That's why we use hashmap for cross referencing.

We also have to make sure that the Palindrome is the longest one.

Finally we store the result and return that.

This explains the first part of our code.

```
hashmap = { }
longest = 0
res = " "
```

Now we need to iterate through the string we have. We do that with a simple for loop.

```
for character in range (len(s)):-
```

Remember the explaination? we have to look left and right from one mid point right?

Here is a catch, If the length of the substring is odd we clearly know where the mid point is, but for even length we have two mid points. We keep that in mind when expanding from centre. Since the substring is expanding dynamically and there could be multiple substring in one string with both odd and even palindrome, that's why we run two for loops.

Don't worry, the code is almost identical. You can copy and paste with one little change.

```
#odd palindrome:- left, right = character, character
# even palindrome:- left, right = character, character + 1 → This is the change.
```

Now we have to expand from center and check if both characters on left and right are same?

while (left ≥ 0 and right ≤ len(s) and s[left] == s[right]) {-

Since we might have multiple palindrome.

stay within index and move left only

stay within index don't exceed array.

are the characters same or not

Great, while loop is good but they are expensive. so we would like to stop if we find longest substring.

If (right - left + 1 > longest) {-

length of current substring

longest = right - left + 1

Update value of longest.

res = [left : right + 1] → store the result.

If current substring is not longest, we are going to expand. remember we are still under the while loop that checks if both characters on left and right are same or not.

left -= 1 ← expand to left

right += 1 → expand to right.

hashmap[s[character]] = character

The character itself

index of that character.

That's it. Remember we need two for loops to tackle both odd and even case.