5. Longest Palindromic Substring
Medium

Given a string s, return the longest palindromic substring in s.

Example 1:

Input: s = "babad"
Output: "bab"
Explanation: "aba" is also a valid answer.

Example 2:

Input: s = "cbbd"
Output: "bb"

```python
from typing import List

class Solution:
    def longestPalindrome(self, s: str) -> str:
        result = ""

        for i in range(len(s)):
            l, r = i, i
            while l >= 0 and r < len(s) and s[l] == s[r]:
                if (r - l + 1) > len(result):
                    result = s[l:r + 1]
                l -= 1
                r += 1

            l, r = i, i + 1
            while l >= 0 and r < len(s) and s[l] == s[r]:
                if (r - l + 1) > len(result):
                    result = s[l:r + 1]
                l -= 1
                r += 1
        return result
```

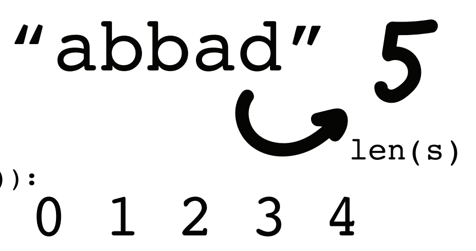
simplified

```python
from typing import List

class Solution:
    def longestPalindrome(self, s: str) -> str:
        result = ""

        for i in range(len(s)):
            for l, r in ((i,i), (i,i+1)):
                while l >= 0 and r < len(s) and s[l] == s[r]:
                    if (r - l + 1) > len(result):
                        result = s[l:r + 1]
                    l -= 1
                    r += 1

        return result
```



`for i in range(len(s)):`

first, check when l and r point to same index

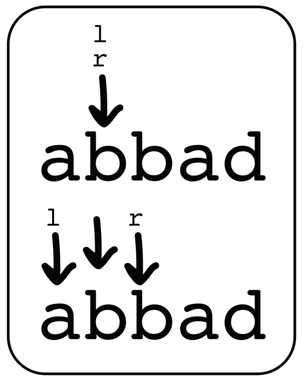
`l, r = i, i`

while l and r pointers are in bound, and s[l] and s[r] are the same

`while l >= 0 and r < len(s) and s[l] == s[r]:`

(r-l+1) is the size of the substring, if (r-l+1) is greater than length of result, then save the new result

```python
        if (r - l + 1) > len(result):
            result = s[l:r + 1]
        l -= 1
        r += 1
```



next, check when l and r point to adjacent index

`l, r = i, i + 1`

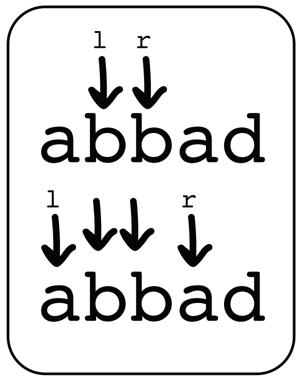
l and r pointers are in bound, and s[l] and s[r] are palindromes

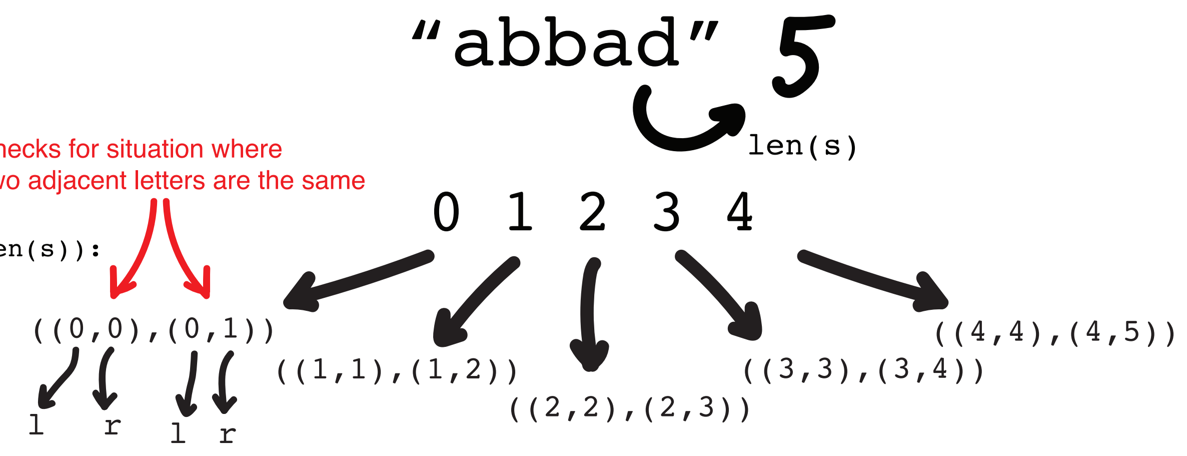
`while l >= 0 and r < len(s) and s[l] == s[r]:`

(r-l+1) is the size of the substring, if (r-l+1) is greater than length of result, then save the new result

```python
    if (r - l + 1) > len(result):
        result = s[l:r + 1]

    l -= 1
    r += 1
```





`for i in range(len(s)):`

`for l, r in ((i,i), (i,i+1)):`

while l and r pointers are in bound, and s[l] and s[r] are the same

`while l >= 0 and r < len(s) and s[l] == s[r]:`

(r-l+1) is the size of the substring, if (r-l+1) is greater than length of result, then save the new result

```python
    if (r - l + 1) > len(result):
        result = s[l:r + 1]

    l -= 1
    r += 1
```

1. Loop through the string, s.
2. There are two parts:
   a. set left (l) and right (r) pointers the same as i, expand left (l-=1) and right (r+=1), and check to see if the s[l] and s[r] are the same.
   b. set left (l) as i, and right (r) as i + 1, so that it accounts for situations where there are two adjacent characters that are the same, and expand left(l-=1) and right (r+=1), and check to see if the s[l] and s[r] are the same.