<u>Problem:</u> Given str, find length of the longest substring without repeating chars

Optimal sol'n: sliding window

```
"abccabb" -> unique: "abc" and "cab" -> longest substr is 3
substring vs. subsequence
"abcbbd" "abccbbd"
"abc" or "abcd" (skips b—repeated char)
sequential
contiguous
```

Intro:

- Verify Constraints
 - substring contiguous? yes (look for substring, not subseq)
 - o case sensitivity? no
- Create Testcases
 - ∘ bestcase: "abccabb" -> 3
 - "cccccc" -> 1
 - · ····· -> ()
 - o overlapping: "abcbda" -> "abc" and "cbda" -> 4

Brute Force:

- Brainstorming & Pattern Observations
- Pseudocode

- Write code
- Run through testcases
- Analyze time and space complexity for brute force:

Time: O(n^2)
Space: O(n)

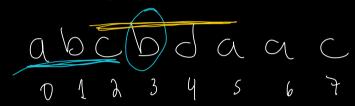
Optimal:
Brainstorming & Pattern Observations

Time: O(n^2)

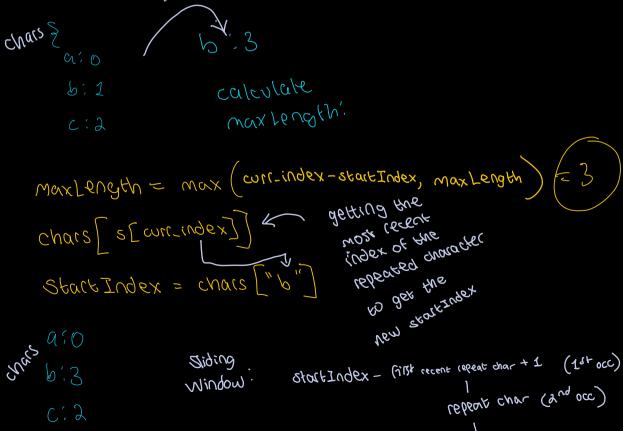
Resetting stacking

Hints:

- use sliding window to represent the current substring
- size of the window will change based on new chars, and chars seen before
- our seen chars hash map keeps track of the index



iterate thru list add each char to the dictionary and its index as its value



· Pseudocode

```
startIndex = 0
curr_index = 0
maxLength = 0
chars = {}

while curr_index < len(s)-maxLength:
    curr_char = s[curr_index]
    //if the current character is a
different repeat
    if curr_char in chars and
chars[curr_char] != curr_index:
        maxLength = max(maxLength,
curr_index-startIndex)
    startIndex = chars[curr_char] + 1
    chars[curr_char] = curr_index
    curr_index= startIndex
    else:
        if curr_char not in chars:
            chars[curr_char] = curr_index
        curr_index += 1</pre>
```

• Write code

• Run through testcases

Analyze time and space complexity

Time: O(n) Space: O(n)