<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

<u>Intro:</u>
- Verify Constraints
  - substring contiguous? yes (look for substring, not subseq)
  - case sensitivity? no
- Create Testcases
  - bestcase: "abccabb" -> 3
  - "cccccc" -> 1
  - "" -> 0
  - overlapping: "abcbda" -> "abc" and "cbda" -> 4

<u>Brute Force:</u>
- Brainstorming & Pattern Observations
- Pseudocode

abcbda

```
while curr index < pointer len(str)          1
                                   keep pointer behind:
                                        prev

    if char not in temp:
      temp += char
      prev += 1
      until get to a repeating char

    else:
      curr.index = prev
      prev--
      temp = " "
```

- Write code
- Run through testcases
- Analyze time and space complexity for brute force:
  - Time: O(n^2) ← resetting starting position
  - Space: O(n)

0 - 10 ← indices
'b' 'b'
iterate from

<u>Optimal:</u>

- Brainstorming & Pattern Observations

  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

```
a  b  c  b  d  a  a  c
0  1  2  3  4  5  6  7
```

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

chars {
- a: 0
- b: 1
- c: 2

b: 3

calculate maxLength:

maxLength = max(curr_index - startIndex, maxLength) ← 3

chars[s[curr_index]] ← getting the most recent index of the repeated character to get the new startIndex

startIndex = chars["b"]

chars
- a: 0
- b: 3
- c: 2

Sliding window: startIndex - first recent repeat char + 1 (1st occ)
|
repeat char (2nd occ)
|

endIndex – third recent char – 1 (3rd occ)

- 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
```

- Write code
- Run through testcases
- Analyze time and space complexity
    Time: O(n)
    Space: O(n)