Problem 2351: First Letter to Appear Twice

Problem Information

Difficulty: Easy
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

Given a string

s

consisting of lowercase English letters, return

the first letter to appear

twice

.

Note

:

A letter

a

appears twice before another letter

b

if the

second

occurrence of

a

is before the

second

occurrence of

b

.

s

will contain at least one letter that appears twice.

Example 1:

Input:

s = "abccbaacz"

Output:

"c"

Explanation:

The letter 'a' appears on the indexes 0, 5 and 6. The letter 'b' appears on the indexes 1 and 4. The letter 'c' appears on the indexes 2, 3 and 7. The letter 'z' appears on the index 8. The letter 'c' is the first letter to appear twice, because out of all the letters the index of its second occurrence is the smallest.

Example 2:

Input:

s = "abcdd"

Output:

"d"

Explanation:

The only letter that appears twice is 'd' so we return 'd'.

Constraints:

2 <= s.length <= 100

s

consists of lowercase English letters.

s

has at least one repeated letter.

Code Snippets

C++:

```
class Solution {
public:
char repeatedCharacter(string s) {

}
};
```

Java:

```
class Solution {
public char repeatedCharacter(String s) {
}
}
```

Python3:

```
class Solution:
def repeatedCharacter(self, s: str) -> str:
```

Python:

```
class Solution(object):
def repeatedCharacter(self, s):
"""
:type s: str
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} s
* @return {character}
*/
var repeatedCharacter = function(s) {

};
```

TypeScript:

```
function repeatedCharacter(s: string): string {

};
```

C#:

```
public class Solution {
public char RepeatedCharacter(string s) {

}
}
```

C:

```
char repeatedCharacter(char* s) {

}
```

Go:

```
func repeatedCharacter(s string) byte {

}
```

Kotlin:

```
class Solution {
fun repeatedCharacter(s: String): Char {

}
}
```

Swift:

```
class Solution {
func repeatedCharacter(_ s: String) -> Character {

}
}
```

Rust:

```
impl Solution {
pub fn repeated_character(s: String) -> char {

}
}
```

Ruby:

```
# @param {String} s
# @return {Character}
def repeated_character(s)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @return String
*/
function repeatedCharacter($s) {

}
}
```

Dart:

```
class Solution {
String repeatedCharacter(String s) {

}
}
```

Scala:

```
object Solution {
def repeatedCharacter(s: String): Char = {

}
}
```

Elixir:

```
defmodule Solution do
@spec repeated_character(s :: String.t) :: char
def repeated_character(s) do

end
end
```

Erlang:

```
-spec repeated_character(S :: unicode:unicode_binary()) -> char().
repeated_character(S) ->
.
```

Racket:

```
(define/contract (repeated-character s)
(-> string? char?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: First Letter to Appear Twice
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
char repeatedCharacter(string s) {

}
};
```

Java Solution:

```
/**
* Problem: First Letter to Appear Twice
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public char repeatedCharacter(String s) {
}
}
```

Python3 Solution:

```
"""
Problem: First Letter to Appear Twice
Difficulty: Easy
Tags: string, hash

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def repeatedCharacter(self, s: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def repeatedCharacter(self, s):
"""
:type s: str
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: First Letter to Appear Twice
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
```

```
/**
* @param {string} s
* @return {character}
*/
var repeatedCharacter = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: First Letter to Appear Twice
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function repeatedCharacter(s: string): string {

};
```

C# Solution:

```
/*
* Problem: First Letter to Appear Twice
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public char RepeatedCharacter(string s) {

}
}
```

C Solution:

```
/*
* Problem: First Letter to Appear Twice
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

char repeatedCharacter(char* s) {

}
```

Go Solution:

```
// Problem: First Letter to Appear Twice
// Difficulty: Easy
// Tags: string, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func repeatedCharacter(s string) byte {

}
```

Kotlin Solution:

```
class Solution {
fun repeatedCharacter(s: String): Char {

}
}
```

Swift Solution:

```
class Solution {
func repeatedCharacter(_ s: String) -> Character {

}
}
```

Rust Solution:

```
// Problem: First Letter to Appear Twice
// Difficulty: Easy
// Tags: string, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn repeated_character(s: String) -> char {

}
}
```

Ruby Solution:

```
# @param {String} s
# @return {Character}
def repeated_character(s)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @return String
*/
function repeatedCharacter($s) {

}
}
```

Dart Solution:

```
class Solution {
String repeatedCharacter(String s) {

}
}
```

Scala Solution:

```
object Solution {
def repeatedCharacter(s: String): Char = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec repeated_character(s :: String.t) :: char
def repeated_character(s) do

end
end
```

Erlang Solution:

```
-spec repeated_character(S :: unicode:unicode_binary()) -> char().
repeated_character(S) ->
.
```

Racket Solution:

```
(define/contract (repeated-character s)
(-> string? char?)
)
```