# Solution Review: Is Unique

This lesson contains the solution review to determine whether a string contains all unique characters or not.

WE'LL COVER THE FOLLOWING



In this lesson, we will consider how to determine if a string has all unique characters. One approach to this problem may be to use an additional data structure, like a hash table. We will also consider how one may solve this problem without the use of such a data structure.

So, we'll present a number of solutions to the problem posed in the previous challenge and give a rough time and space complexity analysis of each approach.

Let's get started!

## Normalization #

First of all, to process the strings, we need to normalize them. The normalization process is as follows:

```
def normalize_str(input_str):
    input_str = input_str.replace(" ", "")
    return input_str.lower()
```

# Solution 1

Now let's discuss the actual implementation. Solution 1 uses a Python dictionary to solve the problem in linear time complexity, but because of the additional data structure, the space complexity is also linear.

## Implementation

Below is the implementation of Solution 1 in Python:

```
def is_unique_1(input_str):
    d = dict()
    for i in input_str:
        if i in d:
            return False
        else:
            d[i] = 1
    return True
```

## Explanation

`d` is initialized to a Python dictionary on **line 2**. Next, there is a `for` loop on **line 3** which iterates `input_str` character by character. In this `for` loop, the condition on **line 4** checks if `i`, i.e., the current character, is present in `d` or not. If it's not present in `d`, the execution jumps to **line** 7 where it is then inserted into `d` as a key with `1` as its value. This is how we record the first instance of any character. However, if `d` is already present in `d`, it means that we encounter it on its second occurrence, which implies that it is not a unique character. Therefore, `False` is returned from the function in case `i` is present in `d`.

If `False` is never returned from the function and the `for` loop terminates, `True` is returned on **line 8**.

I hope everything's been clear up until now!

# Solution 2

Now, solution 2 is very concise and straightforward. In this solution, we make use of `set()`. Let's find out how by having a look at the implementation in

Python.

### Implementation #

```
def is_unique_2(input_str):
    return len(set(input_str)) == len(input_str)
```

### Explanation #

Did you check out how simple the solution looks? Let's discuss it. `set(input_str)` converts `input_str` into a set by removing all the duplicates. Now if the length of that set is equal to the length of `input_str`, it implies that `input_str` did not have any duplicates and has all unique characters. Yes, it is as simple as that!

As we have no idea about how the function `set()` works internally, we cannot be sure about the time and space complexity. However, my understanding of the built-in `set` function is that it is going to take linear time to process all of the elements to determine the set of the list.

## Solution 3 #

It's time for Solution 3. Let's jump to the implementation in Python!

### Implementation #

```
def is_unique_3(input_str):
    alpha = "abcdefghijklmnopqrstuvwxyz"
    for i in input_str:
        if i in alpha:
            alpha = alpha.replace(i, "")
        else:
            return False
    return True
```

### Explanation #

Solution 3 is pretty straightforward. `alpha` is a string defined on **line 2** which contains all 26 letters in lowercase. The `for` loop on **line 3** traverses all the characters in `input_str`. If a character of `input_str`, i.e., `i`, is present in `alpha`, we replace it with an empty string and update `alpha` accordingly on **line 5**. Now as we keep removing `i` in each iteration from `alpha`, if in any iteration we encounter an `i` which is not in `alpha`, it means that it was

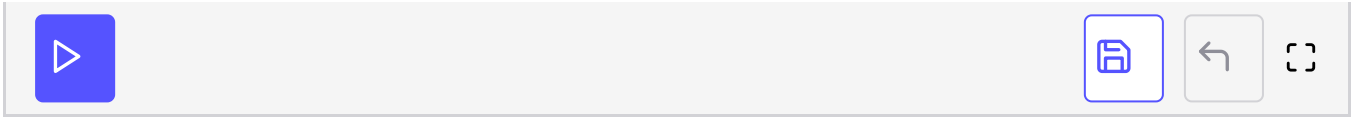
removed in the previous iterations. The execution jumps to **line** 7 and `False` is returned to signal for a duplicate character.

However, if the condition on **line 4** is never `False` in any iteration of the `for` loop, `True` is returned from the function on **line 8** to indicate that `input_str` has all unique characters.

In the code widget below, you can find and execute all three functions that we have discussed above.

```
def normalize_str(input_str):
    input_str = input_str.replace(" ", "")
    return input_str.lower()


def is_unique_1(input_str):
    d = dict()
    for i in input_str:
        if i in d:
            return False
        else:
            d[i] = 1
    return True


def is_unique_2(input_str):
    return len(set(input_str)) == len(input_str)


def is_unique_3(input_str):
    alpha = "abcdefghijklmnopqrstuvwxyz"
    for i in input_str:
        if i in alpha:
            alpha = alpha.replace(i, "")
        else:
            return False
    return True

unique_str = "AbCDefG"
non_unique_str = "non Unique STR"

unique_str = normalize_str(unique_str)
non_unique_str = normalize_str(non_unique_str)
print("Unique String")
print(unique_str)
print("Non-Unique String")
print(non_unique_str, "\n")

print("Solution 1 where input string is unique string")
print(is_unique_1(unique_str))
print("Solution 1 where input string is non-unique string")
print(is_unique_1(non_unique_str), "\n")
```

```
print("Solution 2 where input string is unique string")
print(is_unique_2(unique_str))

print("Solution 2 where input string is non-unique string")
print(is_unique_2(non_unique_str), "\n")

print("Solution 3 where input string is unique string")
print(is_unique_3(unique_str))
print("Solution 3 where input string is non-unique string")
print(is_unique_3(non_unique_str))
```



In the next lesson, we will learn how to convert an integer to a string in Python. See you there!