annand.module07lab01

August 20, 2023

1 Assignment 7

1.0.1 Joseph Annand

1.0.2 8/20/2023

1.1 Question 1

A palindrome is a word, phrase, or sequence that is the same spelled forward as it is backwards. Write a function using a for-loop to determine if a string is a palindrome. Your function should only have one argument.

[5]:
```
# your code here
def for_palindrome(str):
    reverse_list = []
    for i in range(len(str)-1, -1, -1):
        reverse_list.append(str[i])

    reverse_str = "".join(reverse_list)

    return str == reverse_str

print(for_palindrome("racecar"))
print(for_palindrome("chemistry"))
```

```
True
False
```

1.2 Question 2

Write a function using a while-loop to determine if a string is a palindrome. Your function should only have one argument.

[7]:
```
# your code here
def while_palindrome(str):
    str_list = list(str) # or [x for x in str]

    i = 0
    j = len(str_list)-1
    while i < j:
        temp = str_list[i]
        str_list[i] = str_list[j]
        str_list[j] = temp
        i = i + 1
        j = j - 1

    reverse_str = "".join(str_list)

    return str == reverse_str

print(while_palindrome("racecar"))
print(while_palindrome("chemistry"))
```

```
True
False
```

1.3 Question 3

Two Sum - Write a function named two_sum() Given a vector of integers nums and an integer target, return indices of the two numbers such that they add up to target. You may assume that each input would have exactly one solution, and you may not use the same element twice. You can return the answer in any order.Use defaultdict and hash maps/tables to complete this problem.

Example 1: Input: `nums = [2,7,11,15]`, target = 9 Output: `[0,1]` Explanation: Because `nums[0] + nums[1] == 9`, we return `[0, 1]`.

Example 2: Input: `nums = [3,2,4]`, `target = 6 Output: [1,2]`

Example 3: Input: `nums = [3,3]`, `target = 6 Output: [0,1]`

Constraints: `2 <= nums.length <= 104 -109 <= nums[i] <= 109 -109 <= target <= 109` Only one valid answer exists.

[3]:
```
# your code here
from collections import defaultdict

def two_sum(nums, target):
    hash_map = defaultdict(int)
    # Create a key with an empty dictionary as its value to handle repeat values␣
 ↪in nums
    # Use value 119 because it is an impossible key based on constraints
    hash_map[119] = {}
    for i in range(len(nums)):
        # Put repeat values in a nested dictionary to prevent it from␣
 ↪overwriting original entry
        if (target - nums[i]) in hash_map.keys():
            hash_map[119].update({(target - nums[i]): i})
        else:
            # Create key:value pair for non-repeat value
            hash_map[target - nums[i]] = i
    for key, value in hash_map.items():
        comp = (target - key)
        # Search for the complement in the dictionary
        if comp in hash_map.keys():
            # Do not allow the same index to be returned twice
            if comp != key:
                return([value, hash_map[comp]])
            # Search for complement in nested dictionary from repeat values
            else:
                # Search in dictionary with repeat values
                if comp in hash_map[119].keys():
                    return([value, hash_map[119][comp]])


print(two_sum([2,7,11,15], 9))
print(two_sum([3,2,4], 6))
print(two_sum([3,3], 6))
```

```
[0, 1]
[1, 2]
[0, 1]
```

1.4 Question 4

How is a negative index used in Python? Show an example

A negative index is used to access elements in an iterable starting from the end. An index of -1 will access the last element of the iterable, and index of -2 will access the second to last element, and so on.

[4]:
```
# your code here

my_list = [x for x in range(10)]
my_string = "questionfour"

print(my_list)
print(my_list[-2])
print(my_string[-4])
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
8
f
```

1.5 Question 5

Check if two given strings are isomorphic to each other. Two strings str1 and str2 are called isomorphic if there is a one-to-one mapping possible for every character of str1 to every character of str2. And all occurrences of every character in 'str1' map to the same character in 'str2'.

```
Input:  str1 = "aab", str2 = "xxy"
```

```
Output: True
```

```
'a' is mapped to 'x' and 'b' is mapped to 'y'.
```

```
Input:  str1 = "aab", str2 = "xyz"
```

```
Output: False
```

```
One occurrence of 'a' in str1 has 'x' in str2 and other occurrence of 'a' has 'y'.
```

A Simple Solution is to consider every character of ‘str1’ and check if all occurrences of it map to the same character in ‘str2’. The time complexity of this solution is O(n*n).

An Efficient Solution can solve this problem in O(n) time. The idea is to create an array to store mappings of processed characters.

[1]:
```python
# your code here
def check_isomorphic(str1, str2):
    mappings = []

    for i in range(len(str1)):
        if str1[i] in mappings and str2[i] in mappings:
            continue
        elif str1[i] not in mappings and str2[i] not in mappings:
            mappings.append(str1[i])
            mappings.append(str2[i])
        else:
            return False
    return True

print(check_isomorphic("aab", "xxy"))
print(check_isomorphic("aab", "xyz"))
print(check_isomorphic("abc", "xxy"))
```

```
True
False
False
```

[ ]:
```

```