Assignment 7

December 6, 2023

1 Assignment 7

1.0.1 Wanida Ruangsiriluk

1.0.2 December 6, 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.

[1]:
```python
def palindrome(string):
    if (string == string[::-1]):
        return f"{string} is a palindrome"
    else:
        return f"{string} is not a palindrome"


#Test function:
print(palindrome("rotator"))
print(palindrome("yellow"))
```

```
rotator is a palindrome
yellow is not a palindrome
```

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.

[3]:
```python
def palindrome_check(string):
    length = len(string)
    first = 0
    last = length -1
    while first < last:
        if string[first] == string[last]:
            first += 1
            last -= 1
        else:
            return f"{string} is not a palindrome."

        return f"{string} is a palindrome."

print(palindrome_check("level"))
print(palindrome_check("white"))
```

```
level is a palindrome.
white is not a palindrome.
```

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.

[1]:
```
from collections import defaultdict


def two_sum(nums, target):
    indices = defaultdict(list)

    for i, nums in enumerate(nums):
        complement = target - nums

        if complement in indices:
            return indices[complement], [i]
        indices[nums].append(i)
```

[2]:
```
#Example1
nums = [2, 7, 11, 15]
target = 9
result = two_sum(nums, target)
print(result)
```

```
([0], [1])
```

[3]:
```
#Example 2
```

```
nums = [3,2,4]
target = 6
result = two_sum(nums, target)
print(result)
```

```
([1], [2])
```

[4]:
```
#Example 3

nums = [3,3]
target = 6
result = two_sum(nums, target)
print(result)
```

```
([0], [1])
```

[5]:
```
#Example 4, Test code if there is no results
nums = [3,2,4]
target = 10
result = two_sum(nums, target)
print(result)
```

```
None
```

1.4 Question 4

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

Negative index is used to call value of a list or array from the end of a list. The syntax will include -1 as the last index. For example: To print each letter of a word backward, len(word) -1 = n-1 = # of loops from the lenth of a word. The second -1 is a negative or the last index, and the thrid -1 = stepwise backward

[8]:
```
word = "pineapple"
for i in range(len(word) -1, -1, -1):
    print(word[i])
```

```
e
l
p
p
a
e
n
i
p
```

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.

[30]:
```python
str1 = "aab"
str2 = "xxy"


def two_strings(str1, str2):
    if len(str1) != len(str2):
        return False

    mapping_str1 = {}
    mapping_seen = set()

    for i in range(len(str1)):
        str1_char = str1[i]
        str2_char = str2[i]

        if str1_char in mapping_str1:
            if mapping_str1[str1_char] != str2_char:
                return False

        elif str2_char in mapping_seen:
            return False

        mapping_str1[str1_char] = str2_char
        mapping_seen.add(str2_char)

    return True


print(two_strings(str1, str2))
```

```
True
```

[32]:
```
str1 = "aab"
str2 = "xyz"

print(two_strings(str1, str2))
```

```
False
```

[ ]:
```
```