Assignment 7

June 7, 2023

1 Assignment 7

1.0.1 Sue Susman, MEd, BSN, RN

1.0.2 06/06/2023

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]:
```
# your code here
def is_palindrome(string):
    # Remove any whitespace and convert to lowercase
    string = string.replace(" ", "").lower()

    # Iterate over the string using a for-loop
    for i in range(len(string)):
        # Compare characters from both ends of the string
        if string[i] != string[-i - 1]:
            return False

    # If the loop completes without finding any differences, it is a palindrome
    return True
```

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.

[2]:
```
# your code here
string1 = "racecar"
string2 = "Hello"
string3 = "A man a plan a canal Panama"

print(is_palindrome(string1))  # True
```

```
print(is_palindrome(string2))  # False
print(is_palindrome(string3))  # True
```

```
True
False
True
```

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 âĂŞ109 <= nums[i] <= 109 âĂŞ109 <= target <= 109` Only one valid answer exists.

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

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

    # Build a map of numbers and their indices
    for i, num in enumerate(nums):
        num_map[num].append(i)

    # Iterate over the numbers to find the complement
    for i, num in enumerate(nums):
        complement = target - num

        # Check if the complement exists in the map and get its index
        if complement in num_map:
            # If the complement is the same as the current number, check if␣
↪there are at least two occurrences
            if complement == num and len(num_map[complement]) > 1:
                return [num_map[complement][0], num_map[complement][1]]

            # If the complement is different, return the indices of the two␣
↪numbers
            if complement != num:
```

```
                return [i, num_map[complement][0]]

    # No solution found
    return []

# Test cases
nums1 = [2, 7, 11, 15]
target1 = 9
print(two_sum(nums1, target1))  # Output: [0, 1]

nums2 = [3, 2, 4]
target2 = 6
print(two_sum(nums2, target2))  # Output: [1, 2]

nums3 = [3, 3]
target3 = 6
print(two_sum(nums3, target3))  # Output: [0, 1]
```

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

1.4 Question 4

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

[4]:
```
# your code here
my_list = [10, 20, 30, 40, 50]

print(my_list[-1])  # Accessing the last element: 50
print(my_list[-2])  # Accessing the second-to-last element: 40
print(my_list[-3])  # Accessing the third-to-last element: 30
```

```
50
40
30
```

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.

[5]:
```python
# your code here
def is_isomorphic(str1, str2):
    if len(str1) != len(str2):
        return False

    char_map_str1 = {}
    char_map_str2 = {}

    for c1, c2 in zip(str1, str2):
        if c1 in char_map_str1:
            if char_map_str1[c1] != c2:
                return False
        else:
            char_map_str1[c1] = c2

        if c2 in char_map_str2:
            if char_map_str2[c2] != c1:
                return False
        else:
            char_map_str2[c2] = c1

    return True
```

[6]:
```python
str1 = "aab"
str2 = "xxy"
print(is_isomorphic(str1, str2))  # Output: True

str1 = "aab"
str2 = "xyz"
print(is_isomorphic(str1, str2))  # Output: False
```

```
True
False
```