Assignment 7

August 15, 2023

1 Assignment 7

1.0.1 Adeline Casali

1.0.2 August 15, 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]:
```
# Define the function palindrome
def palindrome(string):
    # Convert the input string to lowercase to make the comparison␣
 ↪case-insensitive
    string = string.lower()

    # Initialize a variable to store the reversed version of the input string
    reversed_string = ""

    # Iterate through each character in the input string in reverse order
    for char in reversed(string):
        reversed_string += char

    # Compare the original input string with the reversed_string
    if string == reversed_string:
        return True
    else:
        return False

# Test
print(palindrome("racecar"))
print(palindrome("hello"))
```

```
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.

[3]:
```
# Define the function palindrome
def palindrome(string):
    # Convert the input string to lowercase to make the comparison␣
 ↪case-insensitive
    string = string.lower()

    # Initialize two pointers at beginning and end of the string
    start = 0
    end = len(string) - 1

    # While the start pointer is less than the end pointer
    while start < end:
        # Compare characters at the current positions of start and end pointers
        if string[start] != string[end]:
            return False  # If characters don't match, it's not a palindrome
        start += 1  # Move the start pointer forward
        end -= 1    # Move the end pointer backward

    return True  # If the loop completes without returning False, it's a␣
 ↪palindrome

# Test
print(palindrome("racecar"))
print(palindrome("hello"))
```

```
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.

[5]:
```python
# Import defaultdict
from collections import defaultdict

# Define the function two_sum
def two_sum(nums, target):
    num_indices = defaultdict(list)  # defaultdict to store the indices of each number

    for i, num in enumerate(nums):
        complement = target - num
        if complement in num_indices:
            return [num_indices[complement][0], i]

        num_indices[num].append(i)

# Test
print(two_sum([2, 7, 11, 15], 9))
```

```
[0, 1]
```

1.4 Question 4

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

[8]:
```python
# Negative indexing is used to access elements from the end of a sequence.
# That way, you can access elements in resverse order and not need to know the length of a sequence.

# Example with a list
list = [1, 2, 3, 4, 5]

print(list[-1])
print(list[-2])
print(list[-3])

# Example with a string
string = "Python is cool!"

print(string[-1])
print(string[-5])
```

```
5
4
3
!
c
```

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.

[10]:
```python
# Define the function isomorphic
def isomorphic(str1, str2):
    if len(str1) != len(str2): # Check if lengths of strings are equal
        return False

    mapping = {}  # To store character mappings

    # Iterate through the characters in both strings
    for i in range(len(str1)):
        char1 = str1[i]
        char2 = str2[i]

        # If char1 is already mapped, check if the mapping is consistent with␣
 ↪char2
        if char1 in mapping:
            if mapping[char1] != char2:
                return False
        else:
            # Check if char2 is already used as a mapping for another char in␣
 ↪str1
            if char2 in mapping.values():
                return False
            mapping[char1] = char2

    return True

# Test cases
print(isomorphic("aab", "xxy"))
```

```
print(isomorphic("aab", "xyz"))
```

```
True
False
```