Simonsen.module07lab01

April 28, 2024

1 Assignment 7

1.0.1 Steven Simonsen

1.0.2 4/26/24

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.

[15]:
```
def palindrome(word): #function definition, no default value one string argument
     #code block that converts all string to lower, eliminates space, iterates␣
↪over the word, appends to list
    word=word.lower().replace(" ","")
    forward=[]
    for i in word:
        forward.append(i)
    #If statement compares forward and backwards spelling and prints whether␣
↪word is a palindrome
    if(forward == list(reversed(forward))):
        print('The selected word or phrase is a palindrome')
    else:
        print('The selected word or phrase is not a palindrome')
```

[16]:
```
palindrome(word='Racecar')
```

```
The selected word or phrase is a palindrome
```

[17]:
```
palindrome(word='racecar')
```

```
The selected word or phrase is a palindrome
```

[18]:
```
palindrome(word='Race car')
```

```
The selected word or phrase is a palindrome
```

[19]:
```
palindrome(word='data')
```

```
The selected word or phrase 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.

[1]:
```
def wpalendrome(word): #function definition, no default value one string argument
     #code block converts all string to lower, eliminates space, creates reverse␣
↪version of the string
    clean_word=word.lower().replace(" ","")
    reversed_word=clean_word[::-1]
    #if the forward version doesn't equal reverse version, print no palendrome␣
↪then break to exit loop.
    #Otherwise, print that it is a palendrome
    while clean_word != reversed_word:
        print("The selected word or phrase is not a palindrome")
        break
    else:
        print("The selected word or phrase is a palindrome")
```

[2]:
```
wpalendrome(word='racecar')
```

```
The selected word or phrase is a palindrome
```

[3]:
```
wpalendrome(word='Racecar')
```

```
The selected word or phrase is a palindrome
```

[4]:
```
wpalendrome(word='Race car')
```

```
The selected word or phrase is a palindrome
```

[5]:
```
wpalendrome(word='Steven')
```

```
The selected word or phrase 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.

[27]:
```
list_1 = [2,7,11,15]
target=9
```

[28]:
```
from collections import defaultdict
```

[29]:
```
def two_sum(list_1, target): #function definition, no default value and two␣
↪arguments
    #Code block - assign hashmap to 0, iterate over list. Add value to hash if␣
↪not already present
    hash_map=defaultdict(int)
    work_on = 0
    for i, num in enumerate(list_1):
        work_on += 1
        complement = target - num
        if complement in hash_map:
            #Print complement which is index in hash and i which is orig index.
            print(hash_map[complement],i)
        else:
            hash_map[num]=i
```

[30]:
```
two_sum(list_1, target)
```

```
0 1
```

[32]:
```
list_1 = [3,2,4]
target=6
two_sum(list_1, target)
```

```
1 2
```

[33]:
```
list_1 = [3,3]
target=6
two_sum(list_1, target)
```

```
0 1
```

1.4 Question 4

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

[35]:
```
"""In Python, negative indices are used to genearlly to iterate in reverse order.
↪"""
```

[35]:
```
'In Python, negative indices are used to genearlly to iterate in reverse order.'
```

[6]:
```
"""
Here is an example as used from Question 2 above. The reversed word variable␣
↪uses the syntax [::-1], whichs slices in reverse order.
- The first argument indicates the starting index where the slice begins
```

```
- The second argument indicates the ending index where the slice ends
- The third argument (-1 in this example), is the step size. Typically this is␣
 ↪defaulted to 1, but -1 indicates that we want to reverse the order of the
  iteration sequence.
"""
"""
clean_word=word.lower().replace(" ","")
    reversed_word=clean_word[::-1]
"""
```

[6]:
```
'\nclean_word=word.lower().replace(" ","")\n
reversed_word=clean_word[::-1]\n'
```

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.

[8]:
```
#I did a lot of research on this, and found a very interesting way to solve by␣
 ↪organizing the values into sets, using ord() to convert
#the unicode character into the int equivalent, and then zip to iterate over␣
 ↪each character in both strings and combine into a tuple.
#If the difference is equal to 1, then the values are isomorphic. If greater␣
 ↪than one difference, not isomporphic.
```

[38]:
```
def isomorph(str1, str2): #function definition, no default value and two␣
 ↪arguments
    #Code block 1 - check length of string to ensure equality
    if len(str1) != len(str2):
        return False
```

```
    #Code block 2 - create set from strings, iterate over both using zip and␣
↪each character within strings.
    #Ord converts unicode to integer values. Differences should only be length 1␣
↪if isomorphic
    differences = set(ord(char1) - ord(char2) for char1, char2 in zip(str1,␣
↪str2))
    return len(differences) == 1
```

[39]:
```
print(isomorph("aab", "xxy"))
```

```
True
```

[41]:
```
print(isomorph("aab", "xyz"))
```

```
False
```