Assignment 7

August 18, 2023

1 Assignment 7

1.0.1 Rebecca Hawthorne

1.0.2 8/8/2023

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.

[ ]:
```
def palindrome(word):
    #function to determine if a word is a palindrome
    palindrome_truth = 'is'
    for i in[0, len(word)-1]:
        if (word[i] != word[len(word)-1-i]):
            palindrome_truth = 'is not'
    print ("The word " +word + ' ' +str(palindrome_truth)+ ' a palindrome.')
    return palindrome_truth
```

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.

[ ]:
```
def palindrome_while (word):
    #function to determine if a word is a palindrome using a while loop
    i = 0
    while (word[i] == word[len(word)-1-i]) & (i < (len(word)-1)):
        i +=1
    if i == (len(word)-1):
        print ("The word " +word + ' is a palindrome.')
    else:
        print ("The word " +word + ' is not a palindrome.')

    return
```

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.

I found two options for this function- I did one with two dictionaries and one numerating over a list!

[3]:
```
def two_sums(nums, target):
    #function that checks whether two values in the vector sum to the target.
    #if so, function returns indices of two values.

    #check constraints
    if (2 <= len(nums) <= 104) & (all(-109<= x <= 109 for x in nums))  & (-109␣
↪<= target <= 109):
        from collections import defaultdict
        # Function to return a default
        # values for keys that is not
        # present
        def def_value():
            return "Not Present"

        # Defining the dict
        dif_nums = [target-x for x in nums]
        dif_dict = defaultdict(def_value,␣
↪zip(list(range(0,(len(nums)))),(dif_nums)))

        #look for values in nums that match the values in the dictionary of␣
↪difference values. Also only return if the
        #index of the list is less than the key of the dictionary- this makes␣
↪sure that no number is its own answer,
        #as requested and that each answer is only returned once.
        for index, difference in dif_dict.items():
            for i, val in enumerate(nums) :
                if (val == difference) & (i < index):
                    print ('['+ str(i) + ", " + str(index) +']' )

    else:
```

```
        print('The constraints are not satisfied')


    return
```

[7]:
```
#testing code
print("Test 1:")
nums = [2,7,11,15]
target = 9

two_sums(nums, target)

print("Test 2:")
nums = [3,2,4]
target = 6

two_sums(nums, target)
```

```
Test 1:
[0, 1]
Test 2:
[1, 2]
```

[8]:
```
def two_sums_take_two(nums, target):
    #function that checks whether two values in the vector sum to the target.
    #if so, function returns indices of two values.

    #check constraints
    if (2 <= len(nums) <= 104) & (all(-109<= x <= 109 for x in nums))  & (-109␣
 ↪<= target <= 109):
        from collections import defaultdict
        # Function to return a default
        # values for keys that is not
        # present
        def def_value():
            return "Not Present"

        # Defining the dict for both the nums vector and the difference of nums␣
 ↪and target
        dif_nums = [target-x for x in nums]
        dif_dict = defaultdict(def_value,␣
 ↪zip(list(range(0,(len(nums)))),(dif_nums)))
        nums_dict = defaultdict(def_value,␣
 ↪zip(list(range(0,(len(nums)))),(nums)))
```

```
        #go over both dictionaries and look for common values. Also only return␣
↪if the
        #index of the list is less than the key of the dictionary- this makes␣
↪sure that no number is its own answer,
        #as requested, and that each answer is only returned once.

        for index_dif, difference in dif_dict.items():
            for index_num, val in nums_dict.items():
                if (val == difference) & (index_num < index_dif):
                    print ('['+ str(index_num) + ", " + str(index_dif) +']' )

    else:
        print('The constraints are not satisfied')


    return
```

[9]:
```
#testing code
print("Test 1:")
nums = [2,7,11,15]
target = 9

two_sums_take_two(nums, target)

print("Test 2:")
nums = [3,2,4]
target = 6

two_sums_take_two(nums, target)
```

```
Test 1:
[0, 1]
Test 2:
[1, 2]
```

1.4 Question 4

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

A negative index in Python counts backwards from the end of a list, starting with the last element having an index of -1 and descreasing as you move through the elements to the left.

[2]:
```
#num_list is a list of integers from 1 to 9
num_list = list(range(1,10))

print(num_list)

#then num_list[-1] will be the last element of the list, namely 9
```

```
print(num_list[-1])

#similairly, num_list[-5] would be the fifth from the right or 5

print(num_list[-5])
```

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

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]:
```
def isomorphic(str_1, str_2):

    #inner function that checks if there is a function from one str to the other␣
↪str
    def is_function(str_1, str_2):
        i=0
        is_a_function = True
        pair_list = list(zip(str_1, str_2))
        pair_set=set(pair_list)
        pair_list=list(pair_set)
        pair_list.sort()
        while i<(len(pair_list)-1):
            if (pair_list[i][0]) !=  (pair_list[i+1][0]):
                i +=1
            else:
                is_a_function = False
                break
```

```
        return is_a_function



    #check length of strings to make sure they are equal
    if len(str_1) != len(str_2):
        return "The strings are not the same length and not isomorphic."



    #check if str_1 to str_2 is a function and vice versa
    #if a function and its inverse are both functions, then the function is
    #one-to-one

    else:
        if is_function(str_1, str_2) & is_function(str_2, str_1):
           print(str_1 +" and " + str_2 + " are isomorphic")
        else:
            print (str_1 +" and " + str_2 + " are not isomorphic")

    return
```

[15]:
```
#testing code-these are isomorphic
str_1 = "aab"
str_2 = "xxy"

print(isomorphic(str_1, str_2))

#testing code-these are not isomorphic, there is a function from str_1 to str_2,␣
 ↪but x in str_2 comes from two different
#values in str_1
str_1 = "abc"
str_2 = "xyx"

print(isomorphic(str_1, str_2))

#testing code-these are not isomorphic, a in str_1 goes to different values in␣
 ↪str_2
str_1 = "aab"
str_2 = "xyz"

print(isomorphic(str_1, str_2))

#testing code-these are not isomorphic, different lengths
str_1 = "aabc"
str_2 = "xyz"
```

```
print(isomorphic(str_1, str_2))
```

```
aab and xxy are isomorphic
None
abc and xyx are not isomorphic
None
aab and xyz are not isomorphic
None
The strings are not the same length and not isomorphic.
```