CIND830_Assignment_2.FAGUN

March 22, 2021

0.1 CIND830 - Python Programming for Data Science

0.1.1 Assignment 2 (10% of the final grade)

0.1.2 Due on March 15, 2021 11:59 PM

This is a Jupyter Notebook document that extends a simple formatting syntax for authoring HTML and PDF. Review this website for more details on using Juputer Notebook.

Use the JupyterHub server on the Google Cloud Platform, provided by your designated instructor, for this assignment. Ensure using Python 3.7.6 release then complete the assignment by inserting your Python code wherever seeing the string "#INSERT YOUR ANSWER HERE."

When you click the `File` button, from the top navigation bar, then select `Export Notebook As ...`, a document (PDF or HTML format) will be generated that includes both the assignment content and the output of any embedded Python code chunks.

Using these guidelines, submit **both** the IPYNB and the exported file (PDF or HTML). Failing to submit both files will be subject to mark deduction.

0.1.3 Question 1 [50 pts]:

a) [15 pts] Define a function called `isSymmetrical` that takes a string value as a parameter and returns `TRUE` if the given string is Symmetrical. A string is said to be symmetrical if the reverse of the string is the same as string. For example, "madam" is symmetrical, but "water" is not symmetrical.

[46]:
```
def isSymmetrical(x):
    return x[::-1]

str1 = input("My word:")
str2 = isSymmetrical(str1)

if str1.lower() == str2.lower():
    print ("TRUE")
else:
```

```
    print ("Word isn't symmetrical")
```

```
My word: madam
```

```
TRUE
```

b) [5 pts] Modify the `isSymmetrical` function you created in Q1.a to take any type of data as an input and return `TRUE` if the input is symmetrical. For example, "12321" is symmetrical, but "12345" is not symmetrical.

[47]:
```
def isSymmetrical(x):
    return x[::-1]

value1 = input("Enter the value:")
value2 = isSymmetrical(value1)

if value1 == value2:
    print (value1 + "\tis symmetrical")
else:
    print ( value1 + "\tisn't symmetrical")
```

```
Enter the value: 12345
```

```
12345   isn't symmetrical
```

c) [10 pts] Define a function called `isSymmetricalVec` that calls `isSymmetrical` function you updated in Q1.b to take a list of elements and return their results in a list. For example, given ["1441", "Apple", "radar", "232", "plane"] the function returns [`TRUE`, `FALSE`, `TRUE`, `TRUE`, `FALSE`]

[48]:
```
I = eval(input())

def isSymmetrical(x):
    return x[::-1]

def isSymmetricalVec(l):
    final=[]
    for i in l:
        if isSymmetrical(i) == i:
            j = 'TRUE'
        else:
            j = 'FALSE'
        final.append(j)
    return final

isSymmetricalVec(I)
```

```
["1441", "Apple", "radar", "232", "plane"]
```

[48]: ['TRUE', 'FALSE', 'TRUE', 'TRUE', 'FALSE']

d) [10 pts] Define a function called `isSymmetricalDict` by modifying the function you created in Q1.c to return the results in a dictionary with their corresponding input elements.
For example, given ["1441", "Apple", "radar", "232", "plane"] the function returns {"1441": `TRUE`, "Apple": `FALSE`, "radar": `TRUE`, "232": `TRUE`, "plane": `FALSE`}

[49]:
```
Input = eval(input())

Dict = {}

def isSymmetrical(x):
    return x[::-1]

def isSymmetricalDict(l):
    final=[]
    for i in l:
        if isSymmetrical(i) == i:
            j = 'TRUE'
        else:
            j = 'FALSE'
        final.append(j)
        Dict[i]=j
    return Dict

isSymmetricalDict(Input)
```

```
["1441", "Apple", "radar", "232", "plane"]
```

[49]:
```
{'1441': 'TRUE',
 'Apple': 'FALSE',
 'radar': 'TRUE',
 '232': 'TRUE',
 'plane': 'FALSE'}
```

e) [10 pts] Define a function called `isSymmetricalTuple` by modifying the function you created in Q1.d to traverse the values of the returned dictionary **in reverse order** and return the dictionary keys that has a **`TRUE`** value as a tuple. For example, given ["1441", "Apple", "radar", "232", "plane"] the function returns ("232", "radar","1441").

[50]:
```
Input = eval(input())

Dict = {}
def isSymmetrical(x):
    return x[::-1]
def isSymmetricalTuple(l):
    final=[]
    for i in l:
```

```
        if isSymmetrical(i) == i:
            j = 'TRUE'
        else:
            continue
        final.append(j)
        Dict[i]=j
    return Dict

res = tuple(Dict)


isSymmetricalTuple(Input)
res = tuple(Dict)
print (res[::-1])
```

```
 ["1441", "Apple", "radar", "232", "plane"]
```

```
('232', 'radar', '1441')
```

0.1.4 Question 2 [80 pts] :

a) [15 pts] Define a function called `minSubStr` that takes two string arguments (S and T), and returns `TRUE` if the characters of T exist in S, else returns `FALSE`. For example: given `S = 'xBxxAxxCxxAxCxxBxxxAxCxBxxxAxxBxCx'` and `T = "ABC"`, returns `TRUE` (`T = "ABCD"`, returns `FALSE`)

[53]:
```
def minSubStr(S,T):

    for char in T:
        if char not in S:

            return False

        return True

S = input("Enter a string")
T = input("Enter another a string")
output = minSubStr(S,T)
print (output)
```

```
Enter a string xBxxAxxCxxAxCxxBxxxAxCxBxxxAxxBxCx
Enter another a string ABC
```

```
True
```

b) [15 pts] Modify your `minSubStr` function defined in Q2.a to return the first possible substring (traversing from left) containing all the elements of `T`. For example: given `S =`

'xBxxAxxCxxAxCxxBxxxAxCxBxxxAxxBxCx' and T = "ABC", returns **"BxxAxxC"**

[54]:
```
def minSubStr(S, T):
    for char in T:
        if char not in S:

            return "Not a substring"

    t = list(T)
    start_index = 0
    last_index = 0
    for i in range(len(S)):
        if  S[i] in t:
            t.remove(S[i])
            if len(t) == len(T):
                start_index = i
            if len(t) == 0:
                last_index = i

    return S[start_index + 1: last_index +1 ]
S = input("Enter a string")
T = input("Enter another string")

output = minSubStr(S, T)

print(output)
```

```
Enter a string xBxxAxxCxxAxCxxBxxxAxCxBxxxAxxBxCx
Enter another string ABC

BxxAxxC
```

c) [20 pts] Modify your `minSubStr` function defined in Q2.b to return the shortest substring containing all the elements of T. For example: given `S = 'xBxxAxxCxxAxCxxBxxxAxCxBxxxAxxBxCx'` and `T = "ABC"`, returns **"AxCxB"**

[55]:
```
def minSubStr(S, T):

    substring = []
    s = S
    for i in range(len(s)):
        if s[i] in T:
            if check_substring(s[i:], T):
                substring.append(return_index(s[i:],T))

    return shortest_substring(substring)

def shortest_substring(list_substring):
    shortest_substring = list_substring[0]
```

```
    for i in range(len(list_substring)):
        if len(shortest_substring) > len(list_substring[i]):
            shortest_substring = list_substring[i]
    return shortest_substring

def check_substring(S,T):
     for char in T:
        if char not in S:
            return  False
        return True

def return_index(S, T):
    t = list(T)
    start_index = 0
    last_index = 0
    for i in range(len(S)):
        if  S[i] in t:
            t.remove(S[i])
            if len(t) == len(T):
                start_index = i
            if len(t) == 0:
                last_index = i

    return S[start_index : last_index +1 ]

S = input()
T = input()

output = minSubStr(S, T)

print(output)
```

```
xBxxAxxCxxAxCxxBxxxAxCxBxxxAxxBxCx
ABC
```

```
AxCxB
```

d) [30 pts] modify your `minSubStr` function defined in Q2.c to return the shortest substring containing all the elements of T in order. For example: given `S = 'xBxxAxxCxxAxCxxBxxxAxCxBxxxAxxBxCx'` and `T = "ABC"`, returns **`"AxxBxC"`**

[56]:
```
def minSubStr(S, T):
    substring = []
    s = S
    for i in range(len(s)):
        if s[i] in T:
            if check_substring(s[i:], T):
                substring.append(return_index(s[i:],T))
```

```
    sub = []
    for i in range(len(substring)):
        if substring[i][0] == T[0] and substring[i][-1] == T[-1]:
            sub.append(substring[i])

    return shortest_substring(sub)

def shortest_substring(list_substring):
    shortest_substring = list_substring[0]
    for i in range(len(list_substring)):
        if len(shortest_substring) > len(list_substring[i]):
            shortest_substring = list_substring[i]
    return shortest_substring

def check_substring(S,T):
     for char in T:
        if char not in S:
            return  False
        return True

def return_index(S, T):
    t = list(T)
    start_index = 0
    last_index = 0
    for i in range(len(S)):
        if  S[i] in t:
            t.remove(S[i])
            if len(t) == len(T):
                start_index = i
            if len(t) == 0:
                last_index = i
    return S[start_index : last_index +1 ]

S = input()
T = input()
result = minSubStr(S, T)
print(result)
```

```
xBxxAxxCxxAxCxxBxxxAxCxBxxxAxxBxCx
ABC
```

```
AxxBxC
```

0.1.5 Question 3 [30 pts]:

a) Complete the given code to define three classes, Shape, Oval and Circle. - Shape will be the parent class of Oval and Circle. - Randomly create ten objects from Oval and Circle classes using the `for` loop given below - Your code should produce correct number of Shapes, Circle and Oval objects. The count of objects should be calculated within the class definitions. - Example output (note that since it is random every time it will be a different number of shapes) \ There are 10 random shape objects [‘Circle’, ‘Circle’, ‘Circle’, ‘Circle’, ‘Oval’, ‘Circle’, ‘Oval’, ‘Oval’, ‘Circle’, ‘Circle’] \ 3 of them are ovals \ 7 of them are cirlces \

[1]:
```
class Shape:
    shape_object_counter = 0
    def __init__(self):
        type(self).shape_object_counter +=1

class Oval(Shape):
    oval_object_counter = 0
    def __init__(self):
        type(self).oval_object_counter +=1
        super().__init__()

class Circle(Shape):
    circle_object_counter = 0
    def __init__(self):
        type(self).circle_object_counter +=1
        super().__init__()

from random import choice

list_of_shapes = [choice(['Circle', 'Oval']) for x in range(10)]

for shape in list_of_shapes:
    if shape == "Oval":
        oval = Oval()
    else:
        circle = Circle()

print(list_of_shapes)

print(oval.shape_object_counter + circle.shape_object_counter)

print(oval.shape_object_counter)

print(circle.shape_object_counter)
```

```
['Oval', 'Circle', 'Oval', 'Circle', 'Oval', 'Circle', 'Oval', 'Circle', 'Oval',
'Oval']
10
6
4
```

[ ]:
```
#### This is the end of assignment 2
```