vokfsckfp

July 20, 2023

NAME: PAVITHIRAN. VROLL NO:235229122

Programming and Data Structures with Python Lab3. List Processing in Python

Question 1

Write a function find_average(student) that takes student tuple as input and print student rollno, name, marks and average marks as output.

Test Cases: 1. stud1 = (1, “rex”, 60, 85, 70) find_average(stud1) Modify the above function find_average(student) so that it processes a tuple of tuples. 2. stud2 = (2, “rex”, (80, 75, 90)) find_average(stud2)

[8]:
```
def find_average(stud):
    if len(stud) == 5:
        mark1 = stud[2]
        mark2 = stud[3]
        mark3 = stud[4]
    else:
        mark1 = stud[2][0]
        mark2 = stud[2][1]
        mark3 = stud[2][2]
    print("Roll no: {}".format(stud[0]))
    print("Name : {}".format(stud[1]))
    print("marks 1: {}".format(mark1))
    print("marks 2: {}".format(mark2))
    print("marks 3: {}".format(mark3))
    print("Average marks: {}".format((mark1 + mark2 + mark3)/3))
```

[9]:
```
stud1 = (1, "rex", 60, 85, 70)
find_average(stud1)
```

```
Roll no: 1
Name : rex
marks 1: 60
marks 2: 85
marks 3: 70
Average marks: 71.66666666666667
```

[10]:
```
stud2 = (2, "rex", (80, 75, 90))
find_average(stud2)
```

```
Roll no: 2
Name : rex
marks 1: 80
marks 2: 75
marks 3: 90
Average marks: 81.66666666666667
```

Question 2

Write a weight management program that prompts the user to enter in 7 days of their body weight values as float numbers. Store them in list. Then print first day weight, last day weight, 4th day weight, highest weight, lowest weight and average weight. Finally, print if average weight < lowest weight, then print "Your weight management is excellent". Otherwise print "Your weight management is not good. Please take care of your diet".

[11]:
```
w = list()
for i in range(1, 8):
    val = float(input("Day {}:Enter your weight :".format(i)))
    w.append(val)
print("First day weight: {}".format(w[0]))
print("Last day weight: {}".format(w[-1]))
print("4th day weight: {}".format(w[3]))
print("Highest weight: {}".format(max(w)))
print("Lowest weight: {}".format(min(w)))
avg_w = sum(w)/7
if avg_w < min(w):
    print("Your weight management is excellent")
else:
    print("Your weight management is not good.Please take care of you diet")
```

```
Day 1:Enter your weight :90
Day 2:Enter your weight :89
Day 3:Enter your weight :89
Day 4:Enter your weight :87
Day 5:Enter your weight :88
Day 6:Enter your weight :87
Day 7:Enter your weight :88
First day weight: 90.0
Last day weight: 88.0
4th day weight: 87.0
Highest weight: 90.0
Lowest weight: 87.0
Your weight management is not good.Please take care of you diet
```

Question 3

Write a function lastN(lst, n) that takes a list of integers and n and returns n largest numbers.

How many numbers you want to enter?: 6 Enter a number: 12 Enter a number: 32 Enter a number: 10 Enter a number: 9 Enter a number: 52 Enter a number: 45 How many largest numbers you want to find?: 3 Largest numbers are: 52, 45, 32

[14]:
```
def lastN(n):
    list1 = list()
    for i in range(1, n+1):
        val = int(input("Enter the number: "))
        list1.append(val)
    val2 = int(input("How many largest numbers you want to find?"))
    print("Largest numbers are :", end=" ")
    for y in range(val2):
        temp = max(list1)
        if y == val2-1:
            print(temp)
        else:
            print(temp, end=", ")
        list1.remove(temp)

n = int(input("How many numbers you want to enter?:"))
lastN = lastN(n)
```

```
How many numbers you want to enter?:6
Enter the number: 12
Enter the number: 32
Enter the number: 10
Enter the number: 9
Enter the number: 52
Enter the number: 45
How many largest numbers you want to find?3
Largest numbers are : 52, 45, 32
```

Question 4

Given a list of strings, return a list with the strings in sorted order, except group all the strings that begin with 'x' first. Hint: this can be done by making 2 lists and sorting each of them before combining them.

Test Cases: 1. Input: ['mix', 'xyz', 'apple', 'xanadu', 'aardvark'] Output: ['xanadu', 'xyz', 'aardvark', 'apple', 'mix'] 2. Input: ['ccc','bbb','aaa','xcc','xaa'] Output: ['xaa','xcc','aaa','bbb','ccc'] 3. Input: ['bbb','ccc','axx','xzz','xaa'] Output: ['xaa','xzz','axx','bbb','ccc']

[35]:
```
xlist = list()
def sorting(s):
    for i in s:
```

```
        if i[0]=='x':
            xlist.append(i)
            s.remove(i)
    print(sorted(xlist)+sorted(s))
    del xlist[:]
```

[36]:
```
val = ['min', 'xyz', 'apple', 'xanadu', 'aardvark']
sorting(val)
```

```
['xanadu', 'xyz', 'aardvark', 'apple', 'min']
```

Question 5

Develop a function sort_last(). Given a list of non-empty tuples, return a list sorted in increasing order by the last element in each tuple. Hint: use a custom key= function to extract the last element form each tuple.

Test Cases: 1. Input: [(1, 7), (1, 3), (3, 4, 5), (2, 2)] Output: [(2, 2), (1, 3), (3, 4, 5), (1, 7)] 2. Input: [(1,3),(3,2),(2,1)] Output: [(2,1),(3,2),(1,3)] 3. Input: [(2,3),(1,2),(3,1)] Output: [(3,1),(1,2),(2,3)]

[2]:
```
def sort_last(tupled_list):
    return sorted(tupled_list, key=lambda x: x[-1])

# Test cases
print(sort_last([(1, 7), (1, 3), (3, 4, 5), (2, 2)]))
print(sort_last([(1, 3), (3, 2), (2, 1)]))
print(sort_last([(2, 3), (1, 2), (3, 1)]))
```

```
[(2, 2), (1, 3), (3, 4, 5), (1, 7)]
[(2, 1), (3, 2), (1, 3)]
[(3, 1), (1, 2), (2, 3)]
```

Question6

Other String Functions a) Define a function first() that receives a tuple and returns its first element b) Define a function sort_first() that receives a list of tuples and returns the sorted c) Print lists in sorted order d) Define a function middle() that receives a a tuple and returns its middle element e) Define a functino sort_middle() that receives a list of tuples and returns it sorted using the key middle f) Print the list [(1,2,3), (2,1,4), (10,7,15), (20,4,50), (30, 6, 40)] in sorted order. Output should be: [(2, 1, 4), (1, 2, 3), (20, 4, 50), (30, 6, 40), (10, 7, 15)]

[9]:
```
# function to return the first element
def first(tuple1):
    return tuple1[0]

# function to return the sorted list of tuple based on
 #first element in tuple
```

```
def sort_first(lis_tup):
    return sorted(lis_tup, key = first)

# function to return the list in a sorted order
def sorted_order(lis):
    sorted_list = sort_first(lis)
    return sorted_list

# function to return the middle value from the tuple
def middle(tuple1):
    return tuple1[len(tuple1)//2]

# function to
def sort_middle(lis_tup):
    return sorted(lis_tup, key = middle)
```

[10]:
```
# Test case
tupled_list = [(1,2,3), (2,1,4), (10,7,15), (20,4,50), (30, 6, 40)]
sort_middle(tupled_list)
```

[10]:
```
[(2, 1, 4), (1, 2, 3), (20, 4, 50), (30, 6, 40), (10, 7, 15)]
```

Question 7

Develop a function remove_adjacent(). Given a list of numbers, return a list where all adjacent same elements have been reduced to a single element. You may create a new list or modify the passed in list.

Test Cases: 1. Input: [1, 2, 2, 3] and output: [1, 2, 3] 2. Input: [2, 2, 3, 3, 3] and output: [2, 3] 3. Input: [ ]. Output: [ ]. 4. Input: [2,5,5,6,6,7] Output: [2,5,6,7] 5. Input: [6,7,7,8,9,9] Output: [6,7,8,9]

[19]:
```
def remove_adjacent(lis):
    return sorted(list(set(lis)))
```

[20]:
```
print(remove_adjacent([1, 2, 2, 3]))
print(remove_adjacent([2, 2, 3, 3, 3]))
print(remove_adjacent([]))
print(remove_adjacent([2,5,5,6,6,7]))
print(remove_adjacent([6,7,7,8,9,9]))
```

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

Question 8

Write a function verbing(). Given a string, if its length is at least 3, add 'ing' to its end. Unless it already ends in 'ing', in which case add 'ly' instead. If the string length is less than 3, leave it unchanged. Return the resulting string. So 'hail' yields: hailing; 'swimming' yields: swimmingly; 'do' yields: do.

[39]:
```
def verbing(word):
    if len(word) <= 3:
        return word
    elif word[-1:-4:-1] == "gni":
        word = word + "ly"
        return word
    else:
        return word + "ing"
```

[40]:
```
# test case
print(verbing("swimming"))
print(verbing("hail"))
print(verbing("do"))
```

```
swimmingly
hailing
do
```

Question 9

Develop a function not_bad(). Given a string, find the first appearance of the substring 'not' and 'bad'. If the 'bad' follows the 'not', replace the whole 'not'...'bad' substring with 'good'. Return the resulting string. So 'This dinner is not that bad!' yields: This dinner is good!

[43]:
```
def not_bad(s):
    not_index = s.find('not')
    bad_index = s.find('bad')

    if not_index != -1 and bad_index != -1 and not_index < bad_index:
        return s.replace(s[not_index:bad_index + 3], 'good')
    return s

# Test case
print(not_bad('This dinner is not that bad!'))
```

```
This dinner is good!
```
