German University in Cairo
Faculty of Media Engineering and Technology
Prof. Dr. Slim Abdennadher
Dr. Aysha ElSafty

Introduction to Computer Science, Winter Semester 2016
Practice Assignment 5

Discussion: 5.11.2016 - 10.11.2016

Exercise 5-1

To Be Discussed

Given a list `A` of floating-point numbers, representing students' GPAs, and a bonus mark as inputs from the user, write a Python algorithm that adds the bonus mark to all students' GPAs.

Solution:

```
list_A = eval(input())
bonus = eval(input())
i = 0
n = len(list_A)
while i < n:
␣ list_A[i] = list_A[i] + bonus
␣ i = i + 1

print(list_A )
```

Exercise 5-2

To Be Discussed

The simplest algorithm to search a list of Numbers `N` for a given key Key is to test successively each element.

```
N = eval(input("Enter a list of numbers:"))
m = len(N)
Key = eval(input("Enter a key:"))
i = 0
FOUND = False
while i < m and FOUND == False:
␣ if Key == N[i]:
␣␣ FOUND = True
␣ else:
␣␣ i = i+1
if FOUND == False:
␣ print("Sorry, key is not in the list")
else:
␣ print("Key found")
```

If a list is already stored in increasing order, a modified sequential search algorithm can be used that compares aganist each element in turn, stopping if a list element exceeds the target value.
Write an algorithm for the modified sequential serach.

Solution:

```
N = eval(input("Enter a list of numbers: "))
m = len(N)
key = eval(input("Enter a key: "))
i = 0
FOUND = False
while i < m and FOUND == False and key >= N[i]:
␣ if key == N[i]:
␣␣ FOUND = True
␣ else:
␣␣ i = i+1

if FOUND == False:
␣ print("Sorry, key is not in the list")
else:
␣ print("Key found")
```

Exercise 5-3

Given two lists `A` and `B`, write an algorithm that uses looping to store the sum of the corresponding elements of the lists `A` and `B` in a new list `C`.

Solution:

```
list_A = eval(input())
list_B = eval(input())
list_C = []   # list_C has length = 0
i = 0
n = len(list_A)
while i < n:
␣ s = list_A[i] + list_B[i]
␣ list_C = list_C + s    # append on list_C
␣ i = i + 1

print(list_C)
```

Exercise 5-4

Dice Role

Write an algorithm that prints a list of n dice six-sided rolls.

Solution:

```
import random

n = eval(input())

i = 0
list_C = []
while (i < n):
␣ list_C = list_C+ random.randint(1, 6)
␣ i = i + 1

print(list_C)
```

Exercise 5-5

Write an algorithm to find the maximum value stored in an (unsorted) list `A`.

Solution:

```
list_A = eval(input())
n = len(list_A)
largest_so_far = list_A[0]
i = 1
while (i < n):
␣ if (list_A[i] > largest_so_far):
␣␣ largest_so_far = list_A[i]
␣ i = i + 1

print(largest_so_far)
```

Exercise 5-6 Thousand Numbers

Given a list of non-negative numbers. Write an algorithm to find the number of

- even positive numbers
- odd positive numbers
- Zeros

Additionally, the algorithm should find the sum of

- even positive numbers
- odd positive numbers

Solution:

```
list_A = eval(input())  # preferably large
n = len(list_A)
i = 0
evenCount = 0
oddCount = 0
zeros = 0
evenSum = 0
oddSum = 0
while (i < n):
␣ if (list_A[i] == 0):
␣␣ zeros = zeros + 1
␣ else:
␣␣ if (list_A[i] %2 == 0):
␣␣␣ evenCount = evenCount + 1
␣␣␣ evenSum = (evenSum + list_A[i])
␣␣ else:
␣␣␣ oddCount = oddCount+1
␣␣␣ oddSum = (oddSum + list_A[i])
␣ i = i + 1
print("The number of even numbers is:",evenCount)
print("The sum of even numbers is:",evenSum)
print("The number of odd numbers is:",oddCount)
print("The sum of odd numbers is:",oddSum)
print("The number of zeros is:",zeros)
```

Exercise 5-7 Print Repeated

Write an algorithm that given an **ordered** list of integers `A` prints the elements in the list that are repeated. If some elements occur more than twice, then these elements should be printed only once.

For example, for the list

```
1 1 1 1 4 6 7 7 8
```

your algorithm should print

```
1 7
```

Solution:

-
  ```
  list_A = eval(input())
  n = len(list_A)
  i = 0
  printed = False
  while i < n - 1:
  ␣ if (list_A[i] != list_A[i+1]):
  ␣␣ printed = False
  ␣ else:
  ␣␣ if printed == False:
  ␣␣␣ print(list_A[i])
  ␣␣␣ printed = True
  ␣ i = i + 1
  ```

-
  ```
  A = eval(input())
  num = A[0] - 1
  i = 0
  n = len(A)
  while (i < n-1):
  ␣ if(num != A[i]):
  ␣␣ if (A[i] == A[i+1]):
  ␣␣␣ num = A[i]
  ␣␣␣ print(num)
  ␣␣␣ i = i + 2
  ␣␣ else:
  ␣␣␣ i = i + 1
  ␣ else:
  ␣␣ i = i + 1
  ```

Exercise 5-8 Reverse List

Write an algorithm that reverses the order of elements of the given list.

Solution:

- Reverse in place

```
list_A = eval(input())
n = len(list_A)
i = 0
j = n-1
while(i < n//2):
␣ temp = list_A[i]
␣ list_A[i] = list_A[j]
␣ list_A[j] = temp
␣ i = i+1
␣ j = j-1
print(list_A)
```

- Reverse in new list by appending

```
list_A = eval(input())
n = len(list_A)
list_B = []    # list_B has length = 0
i = 0
j = n-1
while(i < n):
␣ list_B = list_B+ list_A[j]  # append on list_B
␣ i = i+1
␣ j = j-1
print(list_B)
```

- Reverse in new list by iterating over the new list

```
list_A = eval(input())
n = len(list_A)
list_B = [0] * n   # list_B has length = n, all cells have value = 0
j = n-1            # counter for list_A
i = 0              # counter for list_B
while(i < n):
␣ list_B[i] = list_A[j]   # each cell in list_B is now changed
␣ i = i+1
␣ j = j-1
print(list_B)
```

Exercise 5-9

Write an algorithm that given a list of integers `A` moves all even elements in a list of integers to the front of the list and all odd elements to the rear. Hint: you do not have to maintain any order other than all evens appearing before all odds in the list. For example: if the list is of the form [1,4,5,6,2,10] then the algorithm should create a new list of the form [4,6,2,10,5,1] and prints the elements of the resulting list.

Solution:

```
list_A = eval(input())
n = len(list_A)
list_B = [0] * n
i = c = 0
j = n - 1
while(c < n):
␣ if (list_A[c] % 2 == 1):
␣␣ list_B[j] = list_A[c]
␣␣ j = j - 1
␣ else:
␣␣ list_B[i] = list_A[c]
␣␣ i = i + 1
␣ c = c + 1
print(list_B)
```

Exercise 5-10

Write an algorithm that given a list of integers `A` and a number `x` prints the number of occurrences of `x` in the list. In addition, the algorithm should print the positions where x occurs.
For example, if the list is [1, 2, 4, 1, 3] and `x` is 1 then the algorithm should print
1 occurs in the following positions: 0, 3
The number of occurences of 1 is 2
If the list is [1, 2, 4, 1, 3] and `x` is 0 then the algorithm should print
The number of occurences of 0 is 0

Solution:

```
list_A = eval(input())
n = len(list_A)
x = eval(input())
i = 0
occurence = 0
while (i < n):
␣ if (x == list_A[i]):
␣␣ occurence = occurence + 1
␣ i = i + 1

if (occurence == 0):
␣ print("The number of occurences of", x , "is" , 0)
else:
␣ print(x, "occurs in the following positions:")
␣ i = 0
␣ while (i < n):
␣␣ if (x == list_A[i]):
␣␣␣ print(i)
␣␣ i = i + 1
␣ print("The number of occurences of" , x , "is", occurence)
```

Exercise 5-11 String Manipulation - Palindrome

Write an algorithm that determines whether the String the user inputs is a `palindrome` or not. A palindrome is a piece of text that can be read the same way in either direction (left to right and right to left). Examples of palindromes include words such as `racecar` and `noon`.

Solution:

```
x = input()
i = 0
j = len(x) - 1
palindrome = True

while(i < j and palindrome == True):
␣ if(x[i] != x[j]):
␣␣ palindrome = False
␣ i += 1
␣ j -= 1

if(palindrome):
␣ print("The string is a palindrome")
else:
␣ print("The string is not a palindrome")
```

Exercise 5-12 String Manipulation - Run Length

Given a String containing uppercase characters (A-Z), write an algorithm that compresses repeated ‘runs’ of the same character by storing the length of that run.

```
Example:

Input: WWWWWWWWWWWWBWWWWWWWWWWWWBBBWWWWWWWWWWWWWWWWWWWWWWWWBWWWWWWWWWWWWWW
Output: 12W1B12W3B24W1B14W
```

Solution:

```
x = input()
i = 0
count = 1

while (i < (len(x) - 1)):
␣ if(x[i] == x[i+1]):
␣␣ count += 1
␣ else:
␣␣ print(count,x[i],end=" ")
␣␣ count = 1
␣ i +=1
print(count,x[i],end=" ")
```