SSN College of Engineering

Department of Information Technology

UIT2201 — Programming and Data Structures

2022 – 2023

Exercise — 01

April 4, 2023

U. Pranaav | IT-B | 3122225002093

PART-A

1.

I. AIM:

To determine if three numbers a,b,c can satisfy any one of the arithmetic operations involving addition, subtraction, multiplication, division and exponentiation.

Example operations:

a + b = c

a = b – c

a * b = c

The program will return True if even one of the conditions is true.

II. CODE:

```
# -*- coding: utf-8 -*-


"""

This module provides a series of functions that take in 3 numbers
as input and returns a boolean value based on whether any arithmetic
operation can be performed on them as the output. This is a part
of the exercises given under the course UIT2201 (Programming
and Data Structures).

In this source code I have executed my own logic. The code
follows good coding practices.

Your comments and suggestions are welcome.

Created on Wed Apr 05 2023

Revised on Wed Apr 08 2023

Original Author: U. Pranaav <pranaav2210205@ssn.edu.in>

"""
```

```python
#we check the addition operator
def sum_chk(a,b,c):

    """
    Function checks of addition operation is possible on
    given 3 numbers without changing their order. Output
    is given as a boolean [True if possible and False if
    not possible]. Input can be any number.

    The input numbers are not modified and there are no
    side effects.

    args:
        a: the first number.
        b: the second number.
        c: the third number.

    Returns:
        A boolean value based on whether addition is possible
        or not.

    """
    if a == (b+c):
        return True
    elif (a+b) == c:
        return True
    else:
        return False
#end of sum_chk function


#we check the subtraction operator
def sub_chk(a,b,c):

    """
    Function checks of subtraction operation is possible on
    given 3 numbers without changing their order. Output
    is given as a boolean [True if possible and False if
    not possible]. Input can be any number.

    The input numbers are not modified and there are no
    side effects.

    args:
        a: the first number.
        b: the second number.
        c: the third number.

    Returns:
        A boolean value based on whether subtraction is possible
        or not.

    """

    if a == (b-c):
        return True
    elif (a-b) == c:
        return True
    else:
        return False
```

```
#end of sub_chk function


#we check the multiplication operator
def mul_chk(a,b,c):

    """
    Function checks of multiplication operation is possible
    on given 3 numbers without changing their order. Output
    is given as a boolean [True if possible and False if
    not possible]. Input can be any number.

    The input numbers are not modified and there are no
    side effects.

    args:
        a: the first number.
        b: the second number.
        c: the third number.

    Returns:
        A boolean value based on whether multiplication is
        possible or not.

    """

    if a == (b*c):
        return True
    elif (a*b) == c:
        return True
    else:
        return False
#end of mul_chk function


#we check the division operator
def div_chk(a,b,c):

    """
    Function checks of division operation is possible on
    given 3 numbers without changing their order. Output
    is given as a boolean [True if possible and False if
    not possible]. Input can be any number.

    The input numbers are not modified and there are no
    side effects.

    args:
        a: the first number.
        b: the second number.
        c: the third number.

    Returns:
        A boolean value based on whether division is possible
        or not.

    """

    if c!=0 and a == (b/c):
        return True
    elif b!=0 and (a/b) == c:
        return True
```

```
    else:
        return False
#end of div_chk function


#we check the power operator
def pow_chk(a,b,c):

    """
    Function checks of power operation is possible on
    given 3 numbers without changing their order. Output
    is given as a boolean [True if possible and False if
    not possible]. Input can be any number.

    The input numbers are not modified and there are no
    side effects.

    args:
        a: the first number.
        b: the second number.
        c: the third number.

    Returns:
        A boolean value based on whether power operation is
        possible or not.

    """

    if a == (b**c):
        return True
    elif (a**b) == c:
        return True
    else:
        return False
#end of pow_chk function

#function for checking whether 3 numbers satisfy any arithmetic equation
def arithmetic_chk(a,b,c):

    """
    This function takes in 3 numbers and checks whether any
    one of the arithmetic operations is possible on the given
    3 numbers. Input can be any number.

    The input is not modified in any way and there are no side
    effects.

    args:
        a: the first number.
        b: the second number.
        c: the third number.

    Returns:
        A boolean value based on whether any arithmetic operation
        is possible or not.

    """

    overall_answer = False  #initializing a variable for getting final answer

    if sum_chk(a,b,c):
        overall_answer = True
```

```
    if sub_chk(a,b,c):
        overall_answer = True
    if mul_chk(a,b,c):
        overall_answer = True
    if div_chk(a,b,c):
        overall_answer = True
    if pow_chk(a,b,c):
        overall_answer = True

    return overall_answer
#end of function arithmetic_chk


#driver code
if __name__ == "__main__":
    #this part of the code will only be run when the function is called directly
    #it will not be executed when it is imported as a module

    a,b,c = 5,10,15  #issuing random numbers

    print("Value of a:",a,"\nValue of b:",b,"\nValue of c:",c)

    #stating whether any arithmetic operation can be performed
    print("Output of the function is:",arithmetic_chk(a,b,c))
    print()  #for spacing between lines

    a,b,c = -5,-2,10  #initializing negative values

    print("Value of a:",a,"\nValue of b:",b,"\nValue of c:",c)

    #stating whether any arithmetic operation can be performed
    print("Output of the function is:",arithmetic_chk(a,b,c))
    print()

    a,b,c = 7.7,2.2,3.5  #initializing float values

    print("Value of a:",a,"\nValue of b:",b,"\nValue of c:",c)

    #stating whether any arithmetic operation can be performed
    print("Output of the function is:",arithmetic_chk(a,b,c))
    print()

    a,b,c = 0,0,5  #initializing 0 for two variables

    print("Value of a:",a,"\nValue of b:",b,"\nValue of c:",c)

    #stating whether any arithmetic operation can be performed
    print("Output of the function is:",arithmetic_chk(a,b,c))
    print()

    a,b,c = 11,12,15  #initializing an impossible test case

    print("Value of a:",a,"\nValue of b:",b,"\nValue of c:",c)

    #stating whether any arithmetic operation can be performed
    print("Output of the function is:",arithmetic_chk(a,b,c))
    print()
    #end of code
```

III. OUTPUT:

Value of a: 5
Value of b: 10
Value of c: 15
Output of the function is: True

Value of a: -5
Value of b: -2
Value of c: 10
Output of the function is: True

Value of a: 7.7
Value of b: 2.2
Value of c: 3.5
Output of the function is: True

Value of a: 0
Value of b: 0
Value of c: 5
Output of the function is: True

Value of a: 11
Value of b: 12
Value of c: 15
Output of the function is: False

2.

I. AIM:

To write a Python function minmax(data) that takes in a sequence of one or more numbers and returns the smallest and largest numbers in the form of a tuple. We will also display the length of the sequence as well as the number of comparisons taken to obtain the result.

II. CODE:

```
# -*- coding: utf-8 -*-


"""

This module provides a function that takes in a list as an input
and returns a tuple containing length of input, its minimum and
maximum value, along with number of comparisons as the output
without using any inbuilt functions. This is a part of the
excercises given under the course UIT2201 (Programming and Data
```

```
Structures).

In this source code I have executed my own logic. The code
follows good coding practices.

Your comments and suggestions are welcome.

Created on Wed Apr 05 2023

Revised on Wed Apr 10 2023

Original Author: U. Pranaav <pranaav2210205@ssn.edu.in>

"""


import random


#this is the function for finding maximum and minimum value in a sequence
def minmax(data):

    """
    It finds the minimum and maximum value present in the given
    sequence of numbers along with the length of sequence as well
    as number of comparisons that occured in order to obtain the
    final results. The input sequence must have indices for the
    elements to be defined. Further, we assume that all elements
    in the sequence can be compared using standard Python operators.

    The input sequence may be empty, in which case, 'None' is returned.

    The original input is not modified and there are no side effects.

    args:
        data: a sequence of indexed objects that can be compared.

    Returns:
        A tuple containing lenght of sequence,minimum value,maximum
        value and number of comparisons.

    """

    if not data:  #ensuring that input is not an empty list
        return None

    minimum,maximum = data[0],data[0]  #setting initial values
    comparisons = 0
    length = 0

    for i in data:

        length+=1
        comparisons+=2 #there are two comparisons every iteration of the loop

        if i>maximum:  #checking if element is greater than current maximum value
            maximum = i  #if i is greater, then we update value of maximum
        elif i<minimum:  #checking if element is smaller than current minimum value
            minimum = i  #if i is smaller, then we update value of minimum

    return (length,minimum,maximum,comparisons)  #length of tuple will be number of iterations of for loop
#end of function minmax
```

```
#this is a function for generating a list of random numbers
def list_generator(size=1000,low=-10000,high=10000):

    """
    Create and return a list (of len 'size') containing random integers
    in the range from 'low' to 'high'. It requires the random module to
    generate random numbers using randint.

    args:
        size:  Length of the desired random list
        low:  The lower end of the range of numbers
        high:  The higher end of the range of numbers

    returns:
        A new list of given size containing random integers.

    """

    random_list = []
    for count in range(size):
        random_list.append(random.randint(low, high))  #adding random elements to random_list
    return random_list
#end of function list_generator


#driver code
if __name__ == '__main__':
    #this part of the code will only be run when the function is called directly
    #it will not be executed when it is imported as a module

    #stating how the order of output will be
    print("\nOutput will display in the order: Length of data, minimum value of data, "\
          "maximum value of data, number of comparisons\n")

    test_case = []  #empty list test case

    print("Test case is:",test_case)
    print("Output of function:",minmax(test_case))
    print()

    test_case = ()  #empty tuple test case

    print("Test case is:",test_case)
    print("Output of function:",minmax(test_case))
    print()

    test_case = [5]  #single element in a list

    print("Test case is:",test_case)
    print("Output of function:",minmax(test_case))
    print()

    test_case = list_generator(10)  #generating a list of 10 elements including negative numbers

    print("Test case is:",test_case)
    print("Output of function:",minmax(test_case))
    print()

    test_case = list_generator(10,1,10)  #generating a list of 10 positive elements upto 10
```

```
    print("Test case is:",test_case)
    print("Output of function:",minmax(test_case))
    print()

    test_case = list_generator(10,5,5)  #generating a list of 10 same elements

    print("Test case is:",test_case)
    print("Output of function:",minmax(test_case))
    print()

    test_case = tuple(list_generator(10))  #inputting a tuple of 10 random numbers

    print("Test case is:",test_case)
    print("Output of function:",minmax(test_case))
    print()

    test_case = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']  #a list of characters

    print("Test case is:",test_case)
    print("Output of function:",minmax(test_case))
    print()

    test_case = {x for x in range(1,11)}  #generating a set of numbers between 1 and 10

    print("Test case is:",test_case)
    print("Output of function:",minmax(list(test_case)))  #set cannot be indexed, thus it needs to be converted
into list
    print()

    test_case = ["அரவிந்தன்", "அறவள்ளுவன்", "பாரதி", "ஔவையார்", "மணிமேகலை"]  #test case with
tamil characters

    print("Test case is:",test_case)
    print("Output of function:",minmax(test_case))  #set cannot be indexed, thus it needs to be converted into
list
    print()

    #testing the program
    num_trials = random.randint(4, 10)
    for trials in range(num_trials):
        #determining size of list
        trial_size = random.randint(0, 10)

        #generating a list
        trial_list = list_generator(trial_size)

        #getting answer generated by minmax function
        if minmax(trial_list) == None:
            answer = None
        else:
            answer = (minmax(trial_list)[1],minmax(trial_list)[2])

        #checking the correct answer
        if trial_size == 0:
            right_answer = None
        else:
            right_answer = (min(trial_list),max(trial_list))

        print(answer,right_answer)

        if answer != right_answer:
            raise Exception("Function 'minmax' has returned wrong answer.")
```

```
#end of code
```

III. OUTPUT:

Output will display in the order: Length of data, minimum value of data, maximum value of data, number of comparisons

Test case is: []
Output of function: None

Test case is: ()
Output of function: None

Test case is: [5]
Output of function: (1, 5, 5, 2)

Test case is: [7288, -1176, 562, 6457, 731, -5757, -5475, -6923, 4007, -318]
Output of function: (10, -6923, 7288, 20)

Test case is: [2, 6, 3, 7, 10, 5, 1, 4, 4, 6]
Output of function: (10, 1, 10, 20)

Test case is: [5, 5, 5, 5, 5, 5, 5, 5, 5, 5]
Output of function: (10, 5, 5, 20)

Test case is: (8733, -3995, -6546, 5641, -6817, 7180, 2532, -8901, 5481, 64)
Output of function: (10, -8901, 8733, 20)

Test case is: ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']
Output of function: (8, 'A', 'H', 16)

Test case is: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
Output of function: (10, 1, 10, 20)

Test case is: ['அரவிந்தன்', 'அறவள்ளுவன்', 'பாரதி', 'ஔவையார்', 'மணிமேகலை']
Output of function: (5, 'அரவிந்தன்', 'மணிமேகலை', 10)

(-9486, 7456) (-9486, 7456)
(-9797, 6909) (-9797, 6909)
(-2967, 7699) (-2967, 7699)
(-9191, 9309) (-9191, 9309)
(888, 7615) (888, 7615)
(-5240, 6216) (-5240, 6216)
(1214, 1214) (1214, 1214)

(-2991, 2647) (-2991, 2647)

PART-B

3.

I. AIM:

To write a short Python program that takes in a sequence of integer values and determines if there is a distinct pair of numbers in the sequence whose product is odd and then display all the distinct odd product pairs of numbers. We will also display the total number of comparisons required to obtain all the distinct pairs of numbers whose product is odd.

II. CODE:

```
# -*- coding: utf-8 -*-


"""

This module provides a function that takes in a sequence as an
input and returns a list of pairs whose product is odd inside a
tuple as the output without using any inbuilt functions. This is
a part of the exercises given under the course UIT2201 (Programming
and Data Structures).

In this source code I have executed my own logic. The code
follows good coding practices.

Your comments and suggestions are welcome.

Created on Wed Apr 05 2023

Revised on Wed Apr 10 2023

Original Author: U. Pranaav <pranaav2210205@ssn.edu.in>

"""


import random


#creating a function for finding odd product pairs
def oddprod(input_sequence):

    """
    Returns a list of tuples of pairs of numbers whose product is odd as
    well as number of comparisons done in a tuple. It takes in input of a
    sequence of numbers, and indices for the elements are expected to be
    defined. Any duplicate values [(3,7) and (7,3) are considered to be
    the same] will not be displayed.

    If an empty list is given as an input or if list contains only one element,
    then None is given out as output.

    The input sequence is not modified and there are no side effects.

    args:
        num_list=input sequence of indexed numbers
```

```
    Returns:
        A list of tuples which contains pair of numbers from input sequence
        whose product is an odd number and number of comparisons.

    """

    num_list = list(input_sequence)  #generating a list copy of input sequence

    count = 0  #represents the number of comparisons

    count+=1  #since we are checking if the size of list is greater than 1
    if len(num_list)<=1:  #checking if pair of numbers are possible
        return (None,count)  #if size is less than 2, then no pairs are possible

    pair = []  #gives the pair of numbers whose product is odd

    for i in range(len(num_list)):
        for j in range(i+1,len(num_list)):  #i+1 will be first element because we do not need to check for any
previous element
            count+=1
            if (num_list[i]*num_list[j])%2 != 0:  #checking if product is odd
                count+=1
                if num_list[i] != num_list[j]:  #ensuring that both elements are different
                    count+=1
                    if (num_list[i],num_list[j]) not in pair:
                        count+=1
                        if (num_list[j],num_list[i]) not in pair:
                            pair.append((num_list[i],num_list[j]))
                elif num_list.count(num_list[i])>1:
                    count+=1
                    if (num_list[i],num_list[j]) not in pair:  #checking if there are duplicate
                        count+=1
                        if (num_list[j],num_list[i]) not in pair:
                            pair.append((num_list[i],num_list[j]))

    return (pair,count)
#end of function oddprod


#this is a function for generating a list of random numbers
def list_generator(size=1000,low=-10000,high=10000):

    """
    Create and return a list (of len 'size') containing random integers
    in the range from 'low' to 'high'. It requires the random module to
    generate random numbers using randint.

    args:
        size:  Length of the desired random list
        low:  The lower end of the range of numbers
        high:  The higher end of the range of numbers

    returns:
        A new list of given size containing random integers.

    """

    random_list = []
    for count in range(size):
        random_list.append(random.randint(low, high))  #adding random elements to random_list
    return random_list
```

```
#end of function list_generator



#driver code
if __name__ == '__main__':
    #this part of the code will only be run when the function is called directly
    #it will not be executed when it is imported as a module

    test_case = []  #empty list

    print("Test case is:",test_case)
    print("Odd product pairs:",oddprod(test_case)[0])
    print("Number of comparisons:",oddprod(test_case)[1])
    print()  #for spacing between lines

    test_case = [random.randint(-1000,1000)]  #a list with only a single element

    print("Test case is:",test_case)
    print("Odd product pairs:",oddprod(test_case)[0])
    print("Number of comparisons:",oddprod(test_case)[1])
    print()

    test_case = list_generator(10)  #a randomly generated list of 10 elements including negative numbers

    print("Test case is:",test_case)
    print("Odd product pairs:",oddprod(test_case)[0])
    print("Number of comparisons:",oddprod(test_case)[1])
    print()

    test_case = list_generator(10,1,10)  #a randomly generated list of 10 elements with only positive numbers
upto 10

    print("Test case is:",test_case)
    print("Odd product pairs:",oddprod(test_case)[0])
    print("Number of comparisons:",oddprod(test_case)[1])
    print()

    test_case = list_generator(10,5,5)  #a list of 10 elements of same value

    print("Test case is:",test_case)
    print("Odd product pairs:",oddprod(test_case)[0])
    print("Number of comparisons:",oddprod(test_case)[1])
    print()

    test_case = tuple(list_generator(10))  #inputting a tuple value into function

    print("Test case is:",test_case)
    print("Odd product pairs:",oddprod(test_case)[0])
    print("Number of comparisons:",oddprod(test_case)[1])
    print()

    test_case = set(list_generator(10))  #inputting a set value into function

    print("Test case is:",test_case)
    print("Odd product pairs:",oddprod(test_case)[0])
    print("Number of comparisons:",oddprod(test_case)[1])
    print()

    test_case = [4 ,5 ,5 ,3 ,4 ,5 ,3 ,7 ,3 ,5]  #inputting duplicate values in unordered form

    print("Test case is:",test_case)
    print("Odd product pairs:",oddprod(test_case)[0])
```

```
    print("Number of comparisons:",oddprod(test_case)[1])
    print()
    #end of code
```

III. OUTPUT:

Test case is: []
Odd product pairs: None
Number of comparisons: 1

Test case is: [625]
Odd product pairs: None
Number of comparisons: 1

Test case is: [-4640, -8232, 216, -5062, 6117, -5382, -3889, 5379, 7342, -1545]
Odd product pairs: [(6117, -3889), (6117, 5379), (6117, -1545), (-3889, 5379), (-3889, -1545), (5379, -1545)]
Number of comparisons: 64

Test case is: [6, 4, 2, 6, 1, 9, 4, 2, 1, 7]
Odd product pairs: [(1, 9), (1, 1), (1, 7), (9, 7)]
Number of comparisons: 63

Test case is: [5, 5, 5, 5, 5, 5, 5, 5, 5, 5]
Odd product pairs: [(5, 5)]
Number of comparisons: 137

Test case is: (3963, -6661, 2070, 5768, 2990, -3542, -7297, -6021, 2931, 8145)
Odd product pairs: [(3963, -6661), (3963, -7297), (3963, -6021), (3963, 2931), (3963, 8145), (-6661, -7297), (-6661, -6021), (-6661, 2931), (-6661, 8145), (-7297, -6021), (-7297, 2931), (-7297, 8145), (-6021, 2931), (-6021, 8145), (2931, 8145)]
Number of comparisons: 91

Test case is: {-5216, 9442, 2266, -51, 6383, 2800, 243, 756, 1338, 7293}
Odd product pairs: [(-51, 6383), (-51, 243), (-51, 7293), (6383, 243), (6383, 7293), (243, 7293)]
Number of comparisons: 64

Test case is: [4, 5, 5, 3, 4, 5, 3, 7, 3, 5]
Odd product pairs: [(5, 5), (5, 3), (5, 7), (3, 3), (3, 7)]
Number of comparisons: 113

4.

I. AIM:

To implement our own version of the random module function shuffle(data) that accepts a list of elements and randomly reorders the elements so that each possible order occurs with equal

probability. We will implement this by using the random module function randint(a, b) that returns a uniformly random integer from a to b (including both endpoints a and b).

II. CODE:

```
# -*- coding: utf-8 -*-


"""

This module provides a function that takes in a list as an input
and returns a shuffled list as the output without using any inbuilt
functions. This is a part of the exercises given under the course
UIT2201 (Programming and Data Structures).

In this source code I have executed my own logic. The code
follows good coding practices.

Your comments and suggestions are welcome.

Created on Wed Apr 05 2023

Revised on Wed Apr 08 2023

Original Author: U. Pranaav <pranaav2210205@ssn.edu.in>

"""


import random  #we import random module for generating random integers within a given range


#this is a function for shuffling the list
def shuffle_list(data):

    """
    Shuffles the elements present in a list given as an argument
    input sequences is provided in the form of a list. The indices of
    the elements must be defined.

    If an empty list is given as the input, then the same empty list
    is returned as output.

    The input sequence will be modified into the shuffled list.

    args:
        data: an input sequence which needs to be shuffled

    Returns:
        A shuffled sequence

    """

    all_elem_same = True

    for i in data:  #checking if all the elements in list are same
        if i == data[0]:
            continue
        else:
            all_elem_same = False
```

```
            break

    if all_elem_same:  #returning the same sequence if all elements are same
        return data

    temp_list = data.copy()
    for i in range(len(data)):
        a=random.randint(i,len(data)-1)  #random index value generated from i since we have already iterated
through previous indexes
        data[i],data[a]=data[a],data[i]  #interchanging index value with another value

    if temp_list!=data:  #ensuring that list is shuffled
        return data
    else:
        return shuffle_list(data)  #recursively calling until list is shuffled
#end of function shuffle_list


#this is a function for generating a list of random numbers
def list_generator(size=1000,low=-10000,high=10000):

    """
    Create and return a list (of len 'size') containing random integers
    in the range from 'low' to 'high'. It requires the random module to
    generate random numbers using randint.

    args:
        size:  Length of the desired random list
        low:  The lower end of the range of numbers
        high:  The higher end of the range of numbers

    returns:
        A new list of given size containing random integers.

    """

    random_list = []
    for count in range(size):
        random_list.append(random.randint(low, high))  #adding random elements to random_list
    return random_list
#end of function list_generator


#driver code
if __name__ == '__main__':
    #the following code will only be executed if the python file is run directly
    #code will be ignored if it is imported by another python source

    print("Output for list_generator(0):",list_generator(0))  #expected output is an empty list
    print("Output for list_generator(-10)):",list_generator(-10))  #since size is negative, empty list should be
generated
    print("Output for list_generator(1):",list_generator(1))  #generates a list of size 1
    print()  #for spacing

    test_case = []  #empty list

    print("Test case is:",test_case)
    print("Shuffled list is:",shuffle_list(test_case))
    print()

    test_case = list_generator(1)  #a list with only one element
```

```
    print("Test case is:",test_case)
    print("Shuffled list is:",shuffle_list(test_case))
    print()

    test_case = list_generator(10,1,10)  #generating a list of 10 different elements

    print("Test case is:",test_case)
    print("Shuffled list is:",shuffle_list(test_case))
    print()

    test_case = list_generator(10)  #generating a list of 10 different elements including negative numbers

    print("Test case is:",test_case)
    print("Shuffled list is:",shuffle_list(test_case))
    print()

    test_case = list_generator(10,5,5)  #generating a list of 10 same elements

    print("Test case is:",test_case)
    print("Shuffled list is:",shuffle_list(test_case))
    print()

    test_case = [x for x in range(10,0,-1)]  #generating a list of 10 different elements in descending order

    print("Test case is:",test_case)
    print("Shuffled list is:",shuffle_list(test_case))
    print()

    test_case = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']  #a list of characters

    print("Test case is:",test_case)
    print("Shuffled list is:",shuffle_list(test_case))
    print()

    test_case = ["அரவிந்தன்", "அறவள்ளுவன்", "பாரதி", "ஔவையார்", "மணிமேகலை"]  #test case with
tamil characters

    print("Test case is:",test_case)
    print("Shuffled list is:",shuffle_list(test_case))
    print()
    #end of code
```

III. OUTPUT:

Output for list_generator(0): []
Output for list_generator(-10)): []
Output for list_generator(1): [2103]

Test case is: []
Shuffled list is: []

Test case is: [5908]
Shuffled list is: [5908]

Test case is: [3, 8, 10, 7, 7, 6, 3, 5, 7, 4]
Shuffled list is: [7, 3, 7, 10, 8, 7, 4, 5, 3, 6]

Test case is: [-8275, 9628, -9850, -9407, 2175, -414, -9595, 1005, -2952, 1806]
Shuffled list is: [-9595, 1005, -414, 2175, 1806, -8275, -9407, 9628, -9850, -2952]

Test case is: [5, 5, 5, 5, 5, 5, 5, 5, 5, 5]
Shuffled list is: [5, 5, 5, 5, 5, 5, 5, 5, 5, 5]

Test case is: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
Shuffled list is: [2, 5, 3, 4, 9, 1, 10, 7, 8, 6]

Test case is: ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']
Shuffled list is: ['B', 'D', 'H', 'A', 'C', 'E', 'G', 'F']

Test case is: ['அரவிந்தன்', 'அறவள்ளுவன்', 'பாரதி', 'ஔவையார்', 'மணிமேகலை']
Shuffled list is: ['மணிமேகலை', 'அரவிந்தன்', 'ஔவையார்', 'பாரதி', 'அறவள்ளுவன்']

5.

I. AIM:

To calculate the p-norm value of a vector using a function norm(v,p) that returns the p-norm value of a vector v and norm(v) returns the Euclidean norm of v. Euclidean norm is a special case of p = 2 which represents the length of a vector v. The p-norm value is calculated by taking the p^{th} root of the sum of all values in vector v to the p^{th} power.

II. CODE:

```
# -*- coding: utf-8 -*-


"""

This module provides a function that takes in a sequence as an
input and returns a float value as the output without using any
inbuilt functions. This is a part of the exercises given under
the course UIT2201 (Programming and Data Structures).

In this source code I have executed my own logic. The code
follows good coding practices.

Your comments and suggestions are welcome.

Created on Wed Apr 05 2023

Revised on Wed Apr 08 2023

Original Author: U. Pranaav <pranaav2210205@ssn.edu.in>

"""


import random
```

```python
#function for finding p-norm of a vector v
def norm(v,p=2):

    """
    The given function returns the p-norm value of a vector which
    is the pth root of sum of each value in vector v to the pth power.
    It takes in the vector as a sequence of numbers with defined indices
    and the value of p. p has a default value of 2 for euclidean form.

    The function returns 'None' if an empty sequence is given as input.
    The function returns 'None' if value of p is given as 0 or less than 0.

    The input sequence is not modified and there are no side effects.

    args:
        v: represents the vector, takes input as a sequence of indexed objects.
        p: takes a default value of 2 for euclidean form, or takes in input
        in the form of a number.

    Returns:
        A floating point literal which is the p-norm of the input vector.

    """

    if not v:  #checking if vector is empty
        return None

    if p==0:  #checking if value of p is not 0
        return None

    pow_sum = 0  #initial value of total sum for pth power of elements in vector
    for i in v:
        pow_sum+=(i**p)
    return pow_sum**(1/p)  #taking the root of sum of powers to get p-norm
#end of norm function


#this is a function for generating a list of random numbers
def list_generator(size=1000,low=-10000,high=10000):

    """
    Create and return a list (of len 'size') containing random integers
    in the range from 'low' to 'high'. It requires the random module to
    generate random numbers using randint.

    args:
        size:  Length of the desired random list
        low:  The lower end of the range of numbers
        high:  The higher end of the range of numbers

    returns:
        A new list of given size containing random integers

    """

    random_list = []
    for count in range(size):
        random_list.append(random.randint(low, high))  #adding random elements to random_list
    return random_list
#end of function list_generator
```

```
#driver code
if __name__ == "__main__":
    #this part of the code will only be run when the function is called directly
    #it will not be executed when it is imported as a module

    p = 0  #setting value of p=0
    v = list_generator(10)  #generating a random list of 10 elements

    print("Value of p is:",p,"\nValue of v is:",v)
    print("Output of norm(v,p):",norm(v,p))
    print()  #for spacing

    p = random.randint(-5,5)  #setting value of p to random number
    v = ()  #giving empty tuple as input

    print("Value of p is:",p,"\nValue of v is:",v)
    print("Output of norm(v,p):",norm(v,p))
    print()

    p = 0  #setting p to 0
    v = ()  #setting v to empty tuple

    print("Value of p is:",p,"\nValue of v is:",v)
    print("Output of norm(v,p):",norm(v,p))
    print()

    p = random.randint(-10,-1)  #setting a negative value for p
    v = tuple(list_generator(10))  #generating a random tuple of 10 elements

    print("Value of p is:",p,"\nValue of v is:",v)
    print("Output of norm(v,p):",norm(v,p))
    print()

    p = random.randint(1,10)  #setting value of p to random integer
    v = tuple(list_generator(10))  #generating a random tuple of 10 elements

    print("Value of p is:",p,"\nValue of v is:",v)
    print("Output of norm(v,p):",norm(v,p))
    print()

    p = random.random()  #setting value of p to random decimal value
    v = list_generator(10)  #generating a random list of 10 elements

    print("Value of p is:",p,"\nValue of v is:",v)
    print("Output of norm(v,p):",norm(v,p))
    print()

    v = tuple(list_generator(10))  #generating a random tuple of 10 elements

    print("Value of v is:",v)
    print("Output of norm(v) [euclidean norm]:",norm(v))
    print()
    #end of code
```

III. OUTPUT:

Value of p is: 0
Value of v is: [-5087, 9570, -679, -1853, -9079, 548, 5768, 9365, -441, 3432]
Output of norm(v,p): None

Value of p is: -1
Value of v is: ()
Output of norm(v,p): None

Value of p is: 0
Value of v is: ()
Output of norm(v,p): None

Value of p is: -6
Value of v is: (-6606, -5128, -9081, 4084, 3456, 3512, 8723, 2748, -3529, 5285)
Output of norm(v,p): 2480.5778359348396

Value of p is: 2
Value of v is: (8616, -7160, 5014, -3663, 2348, 6877, -2333, -6423, 8195, 3878)
Output of norm(v,p): 18594.61698987102

Value of p is: 0.9432934419325776
Value of v is: [3418, -2962, -4128, 7007, -5852, -5080, 1241, 8588, -3339, -4363]
Output of norm(v,p): (-6209.0946775809525+3823.957898240905j)

Value of v is: (-9545, -8986, 2245, 626, 8974, -2200, -1480, -8764, -2674, 9850)
Output of norm(v) [euclidean norm]: 21114.702697409688