EX - 1: **BASIC PYTHON PROGRAMS**

T Sadakopa Ramakrishnan | IT - B | 3122225002109

Q1. Write a short program that takes as input three integers, a, b, and c, from the console and determines if they can be used in a correct arithmetic formula (in the given order), like “a + b = c”, “a = b − c”, or “a ∗ b = c”. List different types of test cases to verify the correctness of your program.

Aim:

To read 3 inputs from the console and check if they satisfy any one of the arithmetic equations.

Coding:

```
# -*- coding: utf-8 -*-
"""
This module provides a function that returns all possible
arithmetic operations for the given inputs. This is a part
of the exercises given under the course UIT2201 (Programming
and Data Structures).

In this source code I've executed my own logic and may contain
bugs.
The source code has followed good coding practices.

Your comments and suggestions are welcome.

Created on Wed Apr 05 2023

Revised on Wed Apr 08 2023

Original Author: T. Sadakopa Ramakrishnan
<sadakopa2210221@ssn.edu.in>
"""


def checkOPERATION(a,b,c): #defining an user defined function
    '''
        This function takes in three integer inputs a,b and c
```

```
        and returns all the possible arithmetic formulas
following
        a particular order.

        a,b,c : three integers as input

        Returns : All possible arithmetical operations possible
    '''

    #Initialising sum, diff, prod, div, floor_div, exponent in a
variable
    #to compare using if-else conditonal statements

    sum = a + b
    diff = b - c
    prod = a * b
    div = a / b
    floor_div = a // b
    expo = a ** b

    #Initializing output in a variable to append it to a final
ans_list
    #and only print out the elements of ans_list

    sum_output = "a + b = c is possible"
    diff_ouput = "a - b = c is possible"
    prod_output = "a * b = c is possible"
    div_ouptut = "a / b = c is possible"
    floor_div_output = "a // b = c is possible"
    expo_output = "a ** b = c is possible"

    #creating an empty ans_list to append output message
    ans_list = []


    #if statements to check the possible operations
    if sum == c:
        ans_list.append(sum_output)
    if diff == a:
        ans_list.append(diff_ouput)
    if prod == c:
```

```
        ans_list.append(prod_output)
    if div == c:
        ans_list.append(div_ouptut)
    if floor_div == c:
        ans_list.append(floor_div_output)
    if expo == c:
        ans_list.append(expo_output)

    if len(ans_list) > 0:
        for output in ans_list:
            print(output)
    else:
        print("None of the case is possible")

#End of function checkOPERATION()

#Test cases for the following source code:

if __name__ == "__main__":
    #This part of the program will not be executed when the file
is imported.

    #Checking for same value of a,b and c
    checkOPERATION(1,1,1)
    print()

    #Checking for consecutive numbers
    checkOPERATION(5,4,7)
    print()

    #Checking for negative numbers
    checkOPERATION(-1,-5,-6)
    print()

    checkOPERATION(1,2,3)
    print()

    #Checking for all zeroes
    checkOPERATION(0,0,0) #Returns error, ZeroDivisionError:
division by zero
    print()
```

1. Output for a=b=c=1:

```
a * b = c is possible
a / b = c is possible
a // b = c is possible
a ** b = c is possible
```

2. Output for a = 5, b = 4, c = 7:

```
None of the case is possible
```

3. Output for a = -1, b = -5, b = -6:

```
a + b = c is possible
```

4. Output for a = 1, b = 2, c = 3:

```
a + b = c is possible
```

5. Output for a=b=c= 0:

```
ZeroDivisionError: division by zero
```

Q2. Write a short Python function, minmax(data), that takes a sequence of one or more numbers, and returns the smallest and largest numbers, in the form of a tuple of length two. Do not use the built-in functions min or max in implementing your solution. Suppose there are n elements in the input sequence, how many comparisons are done by your function? Run your function for increasing values of n and observe how the number of comparisons is increasing. What can you conclude from this experiment?

Aim:

To write a python program that returns the minimum and maximum values in a sequence. It also returns the number of comparisons that occur before finding the minimum and maximum values.

Coding:

```
# -*- coding: utf-8 -*-
"""
This module provides a function that returns a tuple
containing the minimum and maximum value of a sequence
without using the inbuilt python functions. This is a part
of the exercises given under the course UIT2201 (Programming
and Data Structures).

In this source code I've executed my own logic and may contain
bugs.
The source code has followed good coding practices.

Your comments and suggestions are welcome.

Created on Wed Apr 05 2023

Revised on Wed Apr 09 2023

Original Author: T. Sadakopa Ramakrishnan
<sadakopa2210221@ssn.edu.in>
"""



def minmax(data):
    '''
```

```
    This function takes in a sequence of objects and
    returns a tuple having minimum value and maximum value
    of the given sequence without using inbuilt functions.
    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 the python operators.

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

    data : input sequence

    Returns : a tuple containing min and max value of the input
sequence
    '''
    count = 0
    #finding size of the given sequence
    size = len(data)

    #Returning None for empty input sequence
    if size == 0:
        count += 1
        return None

    #Returning the only element present in a single element
sequence
    if size == 1:
        count += 1
        return (data[0], data[0])

    #Returning tuple using basic if-else statement when only 2
elements are input
    if size == 2:
        count += 1
        if data[0] < data[1]:
            count += 1
            return (data[0], data[1])
        else:
            count += 1
            return (data[1], data[0])
```

```
    #For size of input sequence greater than  2
    else:
        count += 1
        #assigning first element of list as maximum value
        MAXM = data[0]

        #assigning first element of list as minimum value
        MINM = data[0]

        #iterating through data
        for maxm in data:
            #checking if each element is greater than previously
assigned max value
            if maxm > MAXM:
                count += 1
                MAXM = maxm  #if condition holds good


        #iterating through data
        for minm in data:
            #checking if each element is lesser than previously
assigned min value
            if minm < MINM:
                count += 1
                MINM = minm #if condition holds good

        #packing of tuple with minm and maxm value of input
sequence
        tup = (MINM,MAXM)
        print("Number of comparisons:", count)
        #Returning the tuple
        return tup
#End of function minmax()


# We will use the random module to generate an integer within
# a given range under uniform distribution

import random
def createLIST(n,low,high):
  '''
```

```
    This function gets 3 parameters
    n : no of elements in original sequence
    low : lower value for randint range
    high: higher value for randint range
    and returns a sequence which can be used in main function to
perform
    the necessary operation.
  '''
  seq = []
  for i in range(n):
    seq.append(random.randint(low,high))
  return seq
#End of function createLIST()

if __name__ == "__main__":
     #This part of the program will not be executed when the
file is imported.

    data = [] #Empty list test case
    print("Test case is:", data)
    print("Output of the function", minmax(data))
    print()

    data = () #Empty tuple test case
    print("Test case is:", data)
    print("Output of the function", minmax(data))
    print()

    data = [1] #Single element test case
    print("Test case is:", data)
    print("Output of the function", minmax(data))
    print()

    data = createLIST(5,-10,10) #Both positive and negative
numbers test case
    print("Test case is:", data)
    print("Output of the function", minmax(data))
    print()

    data = createLIST(10,0,10)  #only positive numbers test case
    print("Test case is:", data)
```

```
    print("Output of the function", minmax(data))
    print()

    data = createLIST(5,5,5) #same elements test case
    print("Test case is:", data)
    print("Output of the function", minmax(data))
    print()

    data = tuple(createLIST(10,-100,100)) #tuple of random
numbers test case
    print("Test case is:", data)
    print("Output of the function", minmax(data))
    print()

    data = ['A', 'B', 'C', 'D', 'E'] #A list of characters test
case
    print("Test case is:", data)
    print("Output of the function", minmax(data))
    print()

    data = {x for x in range(1,11)} #A set of numbers test case
    print("Test case is:", data)
    print("Output of the function", minmax(data))
    print() #set cannot be indexed, thus it needs to be
converted to a list
```

1. Empty list test case:

```
Test case is: []
Output of the function: None
```

2. Empty tuple test case:

```
Test case is: ()
Output of the function: None
```

3. Single element test case:

```
Test case is: [1]
Output of the function: (1, 1)
```

4. Both positive and negative numbers test case:

```
Test case is: [-8, -9, 5, -4, -3]
Number of comparisons: 3
Output of the function: (-9, 5)
```

5. Only positive numbers test case:

```
Test case is: [10, 10, 0, 7, 8, 0, 5, 5, 9, 0]
Number of comparisons: 2
Output of the function: (0, 10)
```

6. Same elements test case:

```
Test case is: [5, 5, 5, 5, 5]
Number of comparisons: 1
Output of the function: (5, 5)
```

7. Tuple of random numbers test case:

```
Test case is: (-83, 84, -82, -19, -6, -9, 44, -15, 12, 56)
Number of comparisons: 2
Output of the function: (-83, 84)
```

8. A list of characters test case:

```
Test case is: ['A', 'B', 'C', 'D', 'E']
Number of comparisons: 5
Output of the function: ('A', 'E')
```

Q3. Write a short Python function that takes a sequence of integer values and determines if there is a distinct pair of numbers in the sequence whose product is odd. How many pairs do you need to consider, in the worst case, to find the answer?

Aim:

To write a python program that returns the pairs from a sequence with an odd multiple without returning any duplicates or same numbers.

Coding:

```
# -*- coding: utf-8 -*-
"""
This module provides a function that returns distinct pairs of
numbers whose product is an odd number. This is a part
of the exercises given under the course UIT2201 (Programming
and Data Structures).

In this source code I've executed my own logic and may contain
bugs.
The source code has followed good coding practices.

Your comments and suggestions are welcome.

Created on Wed Apr 05 2023

Revised on Wed Apr 08 2023

Original Author: T. Sadakopa Ramakrishnan
<sadakopa2210221@ssn.edu.in>
"""


import random #importing random module for creating a list to be
given as input

def distinctPAIR(seq):
  '''
    This function takes in a sequence as input and
    returns a list of tuple having distinct pairs of
    numbers whose product is odd.
```

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

    seq : input list that can be used to find distinct pairs

    Returns : A list of tuple having distinct pairs of numbers
                whose product is an odd
  '''

  count = 0
  print("Original seq:", seq) #printing original sequence

  odd_seq = [] #creating an empty list to store only odd numbers
from the given seq


  #iterating through given sequence to find odd numbers and
appending to empty list

  for num in seq:
    if num % 2 != 0:
      count += 1
      odd_seq.append(num)

  odd_seq.sort() #useful for one extreme test case

  prod_list = [] #creating an empty list to store the cartesian
product between
                 # next next numbers in the odd seq


  #two for loops to get the cartesian product and appending it
to the empty list

  for i in range(len(odd_seq)):
    for j in range(i+1,len(odd_seq)):
        prod_list.append((odd_seq[i], odd_seq[j]))

  pairs = [] #creating an empty to list to store the non
repeating pairs
```

```
  #for loop followed by an if statement to remove the repeating
pairs
  #and appending to a new list

  for i in prod_list:
    if i not in pairs:
      count += 1
      pairs.append(i)

  #checking if number of pairs is 0, if it is then printing no
distinct pairs
  #else printing out the pairs

  if len(pairs) == 0:
    count += 1
    print("Number of comparisons:", count)
    return "No distinct pairs"
  else:
    count += 1
    print("Number of comparisons:", count)
    return pairs
#End of function distinctPAIR()


def createLIST(n,low,high):
  '''
    This function gets 3 parameters
    n : no of elements in original sequence
    low : lower value for randint range
    high: higher value for randint range
    and returns a sequence which can be used in main function to
perform
    the necessary operation.
  '''
  seq = []
  for i in range(n):
    seq.append(random.randint(low,high))
  return seq
#End of function createLIST()

#Test cases for the following source code:
```

```
if __name__ == '__main__':
    #This part of the program will not be executed when the file
is imported.

    #Finding distinct pairs for positive numbers
    print("Distinct pairs:", distinctPAIR(createLIST(5,10,100)))
    print()

    #Finding distinct pairs for negative numbers
    print("Distinct pairs:",
distinctPAIR(createLIST(10,-100,-10)))
    print()

    #Finding distinct pairs for repeating case
    print("Distinct pairs:", distinctPAIR([1,2,1]))
    print()

    #Finding distinct pairs for both positive numbers and
negative numbers
    print("Distinct pairs:", distinctPAIR(createLIST(5,-10,10)))
    print()

    #Finding distinct pairs for one extreme case
    print("Distinct pairs:", distinctPAIR([3,4,7,5,3]))
    print()
```

1. Finding distinct pairs for positive numbers:

```
Orginal seq: [96, 30, 79, 11, 92]
Number of comparisons: 4
Distinct pairs: [(11, 79)]
```

2. Finding distinct pairs for negative numbers:

```
Orginal seq: [-10, -45, -72, -87, -39, -91, -80, -69, -20, -84]
Number of comparisons: 16
Distinct pairs: [(-91, -87), (-91, -69), (-91, -45), (-91, -39), (-87, -69),
(-87, -45), (-87, -39), (-69, -45), (-69, -39), (-45, -39)]
```

3. Finding distinct pairs for repeating case:

```
Orginal seq: [1, 2, 1]
Number of comparisons: 4
Distinct pairs: [(1, 1)]
```

4. Finding distinct pairs for both positive numbers and negative numbers:

```
Orginal seq: [-6, 7, -4, 10, -3]
Number of comparisons: 4
Distinct pairs: [(-3, 7)]
```

5. Finding distinct pairs for one extreme case:

```
Orginal seq: [3, 4, 7, 5, 3]
Number of comparisons: 9
Distinct pairs: [(3, 3), (3, 5), (3, 7), (5, 7)]
```

Q4. Python's random module includes a function shuffle(data) that accepts a list of elements and randomly reorders the elements so that each possible order occurs with equal probability. The random module includes a more basic function randint(a, b) that returns a uniformly random integer from a to b (including both endpoints). Using only the randint function, implement your own version of the shuffle function.

Aim:

To write a python program that shuffles a list without the usage of shuffle() function.

Coding:

```
# -*- coding: utf-8 -*-
"""
This module provides a function that returns a list in which
elements would be shuffled from main list without using the
random module shuffle function. This is a part
of the exercises given under the course UIT2201 (Programming
and Data Structures).
```

```
In this source code I've executed my own logic and may contain
bugs.
The source code has followed good coding practices.

Your comments and suggestions are welcome.

Created on Wed Apr 05 2023

Revised on Wed Apr 08 2023

Original Author: T. Sadakopa Ramakrishnan
<sadakopa2210221@ssn.edu.in>
"""


import random #importing random module to create elements in
shuffle list
              #between the range of the min and max value of the
entered sequence.


def mySHUFFLE(data ,n):
  '''
        This functions takes in an input tuple that has
        original list and number of elements and
        returns a list containing the same number of
        elements and same elements from the original list
        but shuffled without using shuffle function of
        random module.

        tup : contains original list and length of original list
as elements

        Returns : A shuffled list with elements shuffled from
original list
        '''
  shuffle_list = []
  print("Original list:",data)
  if len(data) == 0:
      return shuffle_list
```

```
  for ele in data:
        if data.count(ele) == len(data):
            shuffle_list = data
            return shuffle_list
        else:
            a = min(data) #1
            b = max(data) #100
            while len(shuffle_list) != n:
                x = random.randint(a,b)
                if x in data:
                    if shuffle_list.count(x) < data.count(x):
                        shuffle_list.append(x)
                else:
                    continue
  if shuffle_list != data:
        print("Shuffled list:", shuffle_list)
  else:
        mySHUFFLE(data,n)


#Getting input from user
def user_input(n, low, high):
    '''
            This function gets 3 parameters
            n : no of elements in original sequence
            low : lower value for randint range
            high: higher value for randint range
            and returns a sequence which can be used in main
function to perform
            the necessary operation.

            Returns: tuple of original list and length of
original list
    '''
    data = []
    for i in range(n):
        data.append(random.randint(low, high))
    return data, n
#End of function user_input()

#Test cases for this source code:
```

```
if __name__ == '__main__':
    #This part of the program will not be executed when the file
is imported.

    #Shuffling of n elements
    mySHUFFLE([1,2,3,4,5,6], 6)
    print()

    #shuffling of empty list
    empty_list = []
    print(mySHUFFLE((empty_list),0)) #Returns an empty list
    print()

    #shuffling a list with same elements
    print(mySHUFFLE([5,5,5,5,5,5], 6)) #Returns the same
original list
    print()

    #Shuffling with one elements
    print(mySHUFFLE([1],1)) #Returns the one element as a list
    print()
```

1. Shuffling of n elements:

```
Original list: [1, 2, 3, 4, 5, 6]
Shuffled list: [4, 6, 1, 2, 3, 5]
```

2. Shuffling of empty list:

```
Original list: []
[]
```

3. Shuffling a list with same elements:

```
Original list: [5, 5, 5, 5, 5, 5]
[5, 5, 5, 5, 5, 5]
```

4. Shuffling with one elements:

```
Original list: [1]
[1]
```

Q5. The p-norm of a vector v = (v1, v2, · · · , vn) in n-dimensional space is defined as

$$\|v\| = \sqrt[p]{v_1^p + v_2^p + \cdots + v_n^p}$$

For the special case of p = 2, this results in the traditional Euclidean Norm, which represents the length of the vector. Give an implementation of a function named 'norm' such that norm(v, p) returns the p-norm value of v and norm(v) returns the Euclidean norm of v. You may assume that v is a tuple of numbers.

Aim:

To write a python program that returns the P-Norm or Euclidean Norm of a list of elements.

Coding:

```
# -*- coding: utf-8 -*-
"""
This module provides a function that returns p - norm value when
function is called in certain way and Euclidean value when
function is called in the other way. This is a part
of the exercises given under the course UIT2201 (Programming
and Data Structures).

In this source code I've executed my own logic and may contain
bugs.
The source code has followed good coding practices.

Your comments and suggestions are welcome.
```

```
Created on Wed Apr 05 2023

Revised on Wed Apr 08 2023

Original Author: T. Sadakopa Ramakrishnan
<sadakopa2210221@ssn.edu.in>
"""



def norm(v,p=2):
    '''
        This function takes in a tuple of values as v - vector
        and returns the p-norm value.

        v : tuple containing vectors
        p = 2 : default argument is p = 2, euclidean norm

        Returns: p-norm value when function norm(v,p) is called
                 euclidean norm when function norm(v) is called
    '''

    sum = 0 #initializing sum = 0

    #iterating through the vectors and summing up according to
value
    # of p
    for num in v:
        sum = sum + num**p

    #root value is 1 / p
    root = 1 / p

    #finally answer is sum power root
    ans = sum ** root

    #returning the function
    return ans

#End of function norm
```

```
#Test cases for this source code

if __name__ == "__main__":
    #This part of the program will not be executed when the file
is imported.

    #Finding Euclidean norm value
    print("Euclidean norm value", norm((1,2,3,4)))
    print()

    #Finding p - norm value
    print("p - norm value", norm((1,2,3,4,5), 5))
    print()

    #Finding p - norm value for negative value of p
    print("p - norm value", norm((1,2,3,4,5), -2))
    print()
    #Finding p - norm value for p = 0
    print("p - norm value", norm((1,2,3,4,5), 0)) #Returns
ZeroDivisionError: division by zero
    print()
```

1. Finding Euclidean norm value:

```
Euclidean norm value 5.477225575051661
```

2. Finding p - norm value:

```
p - norm value 5.360220495669696
```

3. Finding p - norm value for negative value of p:

```
p - norm value 0.8265842980736917
```

4. Finding p - norm value for p = 0:

```
ZeroDivisionError: division by zero
```