Ex. No: 1 11-04-2023	UIT2201 — Programming and Data Structures

Aim:

To execute the following programs and note the output.

PART – A

1. 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.

Code:

```
"""

    This module provides a function that returns
    a string object, which is an expression where
    the inputs a,b and c satisfy the returned expression.

    It also provides with another function which is
    used to test the checkarithmetic() function.

    Original Author : Pranesh Kumar

    Created On Sat 08 Apr 2023
"""

# We will use the random module to generate
# random values to test the function
import random


def checkarithmetic(a: int, b: int, c: int) -> str:
    """
        Checks whether the given set of inputs satisfy the specified
expressions and
        returns the expression as string

        Inputs:
            3 integers a,b and c
        Returns:
            Expression which satisfies the given inputs.
            If no expression is satisfied, "None" is returned
            If the 3 inputs are 0, "All expressions" is returned

    """
    if a == b == c == 0:
        return "All expressions"
    elif a + b == c:
        return "a + b = c"
```

```
    elif a + c == b:
        return "a + c = b"
    elif b + c == a:
        return "b + c = a"
    elif a == b - c:
        return "a = b - c"
    elif a == c - b:
        return "a = c - b"
    elif b == a - c:
        return "b = a - c"
    elif b == c - a:
        return "b = c - a"
    elif c == a - b:
        return "c = a - b"
    elif c == b - a:
        return "c = b - a"
    elif a * b == c:
        return "a * b = c"
    elif a * c == b:
        return "a * c = b"
    elif b * c == a:
        return "b * c = a"
    else:
        return "None"


# End of func checkarithmetic()


def testfunction(nooftestcases: int, startvalue: int, endvalue: int) ->
None:
    """
        Tests the checkarithmetic() function by providing random values for
        a,b and c within the limits startvalue and endvalue.

        Stores the result in a dictionary, where the key is a tuple
        of values of a,b and c and the value is the expression returned by
the function

        startvalue and endvalue specifies the range of random values to be
generated
        nooftestcases refers to the number of testcases to be tested

        Inputs:
            nooftestcases : int
            startvalue : int
            endvalue : int

        Returns:
            None

    """
    testcases: dict = {}
    for _ in range(nooftestcases):
        a: int = random.randint(startvalue, endvalue)
        b: int = random.randint(startvalue, endvalue)
        c: int = random.randint(startvalue, endvalue)
        testcases[(a, b, c)] = checkarithmetic(a, b, c)

    print("The testcases are:")
    for key, value in testcases.items():
```

```
        print(f"{key} : {value}")


# End of func testfunction()


# Testing code
if __name__ == "__main__":
    # Following code will be executed only when this Python
    # file is run directly.  Code will be ignored if this
    # file is imported by another Python source.

    # using our function to test the checkarithmetic() func
    testfunction(nooftestcases=20,
                 startvalue=0,
                 endvalue=20)

    # manually entering values for a, b and c and then testing
    num1: int = int(input("Enter the value for A: "))
    num2: int = int(input("Enter the value for B: "))
    num3: int = int(input("Enter the value for C: "))
    print(checkarithmetic(num1, num2, num3))
```

Inputs and Output:

```
The testcases are:
(4, 2, 10) : None
(1, 10, 4) : None
(16, 9, 20) : None
(14, 12, 18) : None
(8, 5, 1) : None
(20, 17, 19) : None
(14, 3, 16) : None
(18, 3, 8) : None
(6, 15, 16) : None
(3, 14, 11) : a + c = b
(10, 14, 19) : None
(19, 16, 8) : None
(10, 11, 18) : None
(10, 18, 11) : None
(13, 19, 12) : None
(18, 10, 14) : None
(6, 9, 2) : None
(4, 2, 2) : b + c = a
(2, 20, 10) : a * c = b
(6, 1, 0) : None
Enter the value for A: 35
Enter the value for B: 19
Enter the value for C: 16
b + c = a
```

Explanation:

The code defines a function checkarithmetic() that takes three integer inputs and checks whether they satisfy any of the given expressions. It returns the expression as a string, or "None" if none of the expressions are satisfied. The code also provides a testing function testfunction() that generates random values for a, b, and c and tests the checkarithmetic() function. The testing function stores the test results in a dictionary and prints them. The code then allows the user to manually input values for a, b, and c to test the checkarithmetic() function.

2. 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?

Code:

```
"""

    This module provides a function that returns
    a tuple object, which contains the minimum element
    followed by the maximum element in the given sequence

    It also provides with another function which is
    used to test the finminmax() function.

    Original Author : Pranesh Kumar

    Created On Sat 08 Apr 2023
"""

# We will use the random module to generate
# random values to test the function
import random


def findminmax(seq: list) -> tuple:
    """This function returns a tuple which contains the minimum element
    followed by the maximum element.
    Returns a tuple with None, when the sequence is empty.
    Returns a tuple with same values, when the sequence is singleton.


    Args:
        seq (list): input sequence which can be any indexed iterable,
                        preferably a list

    Returns:
        tuple: tuple which contains the minimum element followed by the
maximum
```

```
                element

    """
    seqsize = len(seq)
    comparisonCount: int = 0
    if seqsize == 0:
        return None, comparisonCount
    elif seqsize == 1:
        return seq[0], seq[0], comparisonCount
    elif seqsize == 2:
        if seq[0] < seq[1]:
            return seq[0], seq[1], comparisonCount
        else:
            return seq[1], seq[0], comparisonCount
    else:
        minelt = seq[0]
        maxelt = seq[0]
        for idx in range(1, seqsize):
            if seq[idx] > maxelt:
                comparisonCount += 1
                maxelt = seq[idx]
            elif seq[idx] < minelt:
                comparisonCount += 1
                minelt = seq[idx]
        return minelt, maxelt, comparisonCount


# End of findminmax() func

def createrandomlist(size: int = 1000, startvalue: int = -10000, endvalue:
int = 10000) -> list:
    """This function creates a list of specified size with values ranging
from
    the specified start and end values

    Args:
        size (int, optional): length of the required list. Defaults to
1000.
        startvalue (int, optional): starting range for generating random
value. Defaults to -10000.
        endvalue (int, optional): ending range for generating random value.
Defaults to 10000.

    Returns:
        list: list with the required number of values within the specified
range
    """
    randomlist: list = []
    for count in range(size):
        randomlist.append(random.randint(startvalue, endvalue))
    return randomlist


# End of createrandomlist() func


def testfunction(nooftestcases: int) -> None:
    """This function tests the findminmax() function, by testing it with
    random lists with the help of createrandomlist() function for specified
number
    of times.
```

```
    Also checks if the returned value matches with the value generated with
    the help of built-in functions. If it does not match, it prints a
    message.

    Args:
        nooftestcases (int): number of testcases to be tested
    """
    for count in range(nooftestcases):
        mySize = count + (10 * (count + 1 ))
        testlist = createrandomlist(size = mySize)
        answer: tuple = findminmax(testlist)
        rightanswer: tuple = min(testlist), max(testlist)
        print(mySize, answer, rightanswer)
        if answer[0:2] != rightanswer:
            print("Function has returned a wrong answer!")


# End of testfunction() func


# Testing code
if __name__ == "__main__":
    """Following code will be executed only when this Python
    file is run directly.  Code will be ignored if this
    file is imported by another Python source
    """

    # Manual testing
    print(findminmax([]))
    print(findminmax([100]))
    try:
        print(findminmax({1, 2, 3, 4}))  # sets does not work with our
function
    except Exception as e:
        print(e)
    finally:
        print(findminmax("Hello!"))

    # using our function to test
    testfunction(nooftestcases=10)
```

Output:

```
(None, 0)
(100, 100, 0)
'set' object is not subscriptable
('!', 'o', 4)
10 (-9342, 7228, 3) (-9342, 7228)
21 (-9666, 9705, 6) (-9666, 9705)
32 (-9608, 9946, 9) (-9608, 9946)
43 (-9399, 8165, 6) (-9399, 8165)
54 (-9835, 9508, 6) (-9835, 9508)
65 (-9676, 9940, 5) (-9676, 9940)
76 (-9436, 9212, 5) (-9436, 9212)
87 (-9946, 9936, 7) (-9946, 9936)
98 (-9600, 9991, 9) (-9600, 9991)
109 (-9894, 9917, 14) (-9894, 9917)
```

Explanation:

This program defines two functions: "findminmax(seq)" and "createrandomlist(size, startvalue, endvalue)". The findminmax function takes a list as input and returns a tuple containing the minimum and maximum elements of the list, along with the number of comparisons made during the operation. The createrandomlist function creates a list of random values within a specified range. The program also defines a third function, "testfunction(nooftestcases)", which tests the findminmax function using random lists generated by createrandomlist. The testfunction compares the result of findminmax with the result generated by Python's built-in min and max functions. The program also includes some manual testing of the findminmax function. Finally, the program executes the testfunction 10 times and prints the test results.

For increasing values on n, the number of comparisons increases. This may not be completely noticeable in the above program, but for large values of n, the number of comparisons increases significantly.

PART – B

3. 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?

Coding:

```
"""
    This module provides a function checkproduct() that returns a tuple
object
    which consists of:
        1. count - refers to the number of combinations required to produce
an odd product
        2. tuple - refers to the tuple that contains the elements which are
multiplied
        3. product - refers to the product after multiplying the numbers in
the tuple

    It also consists of a function that tests our function, with random
lists and
    checks it with the output produced with the help of built-in functions.

    Original Author : Pranesh Kumar

    Created On Sat 08 Apr 2023
"""

# We will use the random module to generate
# random values to test the function
import random

# We will use the itertools module to test the output produced
# by our function
```

```
from itertools import product


def checkoddproduct(seq: iter) -> tuple[int, tuple, int]:
    """This function provides a way to check the number of combinations or
pairs
    required to produce an odd product.

    Args:
        seq (iter): can be any indexed iterable

    Returns:
        tuple[int, tuple, int]: tuple consisting of 3 elements:
        1. count - number of pairs required
        2. tuple - numbers in the pair
        3. product - product of the numbers in the pair
    """
    count: int = 0
    for i in seq:
        for j in seq:
            count += 1
            if (i * j) % 2 != 0:
                return count, (i, j), i * j


# end of checkoddproduct() func


def createrandomlist(size: int = 10, startvalue: int = -10000, endvalue:
int = 10000) -> list:
    """This function creates a list of specified size with values ranging
from
    the specified start and end values

    Args:
        size (int, optional): length of the required list. Defaults to 10.
        startvalue (int, optional): starting range for generating random
value. Defaults to -10000.
        endvalue (int, optional): ending range for generating random value.
Defaults to 10000.

    Returns:
        list: list with the required number of values within the specified
range
    """
    randomlist: list = []
    for count in range(size):
        randomlist.append(random.randint(startvalue, endvalue))
    return randomlist


# End of createrandomlist() func


def testfunction(nooftestcases: int) -> None:
    """This function tests the checkoddprod() function, by testing it with
    random lists with the help of createrandomlist() function for specified
number
    of times.

    Also checks if the returned value matches with the value generated with
```

```
    the help of built-in functions. If it does not match, it prints a
    message.

    Args:
        nooftestcases (int): number of testcases to be tested
    """
    for count in range(nooftestcases):
        testlist = createrandomlist()
        answer: tuple = checkoddproduct(testlist)
        combinedseq = list(product(testlist, testlist))
        rightanswer = ()
        for idx, elt in enumerate(combinedseq):
            if (elt[0] * elt[1]) % 2 != 0:
                rightanswer: tuple = idx + 1, (elt[0], elt[1]), elt[0] *
elt[1]
                break
        print(answer, rightanswer, sep=" -- ")
        if answer != rightanswer:
            print("Function has returned a wrong answer!")


# End of testfunction() func


if __name__ == "__main__":
    """Following code will be executed only when this Python
    file is run directly.  Code will be ignored if this
    file is imported by another Python source
    """

    # Manual testing
    print(checkoddproduct([1, 6, 4, 7, 8]))
    print(checkoddproduct([2, 4, 6, 8, 10]))
    try:
        print(checkoddproduct(['ABCDE'])) # strings won't work here
    except Exception as e:
        print(e)
    print(checkoddproduct((10, 3, 45, 98, 100)))
    print(checkoddproduct({10, 3, 45, 98, 100})) # sets would work

    # using our function to test
    testfunction(nooftestcases=15)
```

Output:

```
(1, (1, 1), 1)
None
can't multiply sequence by non-int of type 'str'
(7, (3, 3), 9)
(7, (3, 3), 9)
(34, (2613, 2613), 6827769) -- (34, (2613, 2613), 6827769)
(12, (-4097, -4097), 16785409) -- (12, (-4097, -4097), 16785409)
(1, (9515, 9515), 90535225) -- (1, (9515, 9515), 90535225)
(12, (4335, 4335), 18792225) -- (12, (4335, 4335), 18792225)
(34, (-4495, -4495), 20205025) -- (34, (-4495, -4495), 20205025)
(1, (359, 359), 128881) -- (1, (359, 359), 128881)
(12, (-6479, -6479), 41977441) -- (12, (-6479, -6479), 41977441)
(34, (1575, 1575), 2480625) -- (34, (1575, 1575), 2480625)
(12, (1323, 1323), 1750329) -- (12, (1323, 1323), 1750329)
(1, (-8317, -8317), 69172489) -- (1, (-8317, -8317), 69172489)
(1, (-6007, -6007), 36084049) -- (1, (-6007, -6007), 36084049)
(12, (4633, 4633), 21464689) -- (12, (4633, 4633), 21464689)
(12, (3383, 3383), 11444689) -- (12, (3383, 3383), 11444689)
(12, (5003, 5003), 25030009) -- (12, (5003, 5003), 25030009)
(78, (-4807, -4807), 23107249) -- (78, (-4807, -4807), 23107249)
```

Explanation:

This is a Python module that contains a function named checkoddproduct() which takes an iterable as an input and returns a tuple of three values - count, tuple, and product. The count indicates the number of pairs required to produce an odd product. The tuple contains the pair of elements that are multiplied to produce the odd product. The product is the odd product itself. The module also has a testing function named testfunction() that tests the checkoddproduct() function with random lists and checks the output produced by it with the help of built-in functions. If the output does not match, it prints an error message. The module also contains two utility functions - createrandomlist() and testfunction() that create random lists and test the checkoddproduct() function respectively. The main code block tests the checkoddproduct() function manually and also by calling the testfunction() function.

For the worst-case scenario, if the list contains n number of elements, then we would require n^2 comparisons would be required.

5. 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.

Coding:

```
"""
        This module provides a function shuffleseq() which shuffles
        an iterable in such a way that each possible order occurs with
equal probability.

        It also provides with another function for testing our function,
which verifies
        whether the given sequence is shuffled maintaining its elements and
its length.

        Original Author : Pranesh Kumar

        Created On Sat 08 Apr 2023
"""

# We will use the random module to generate
# random indices to test the function
import random


def shuffleseq(data: iter) -> list:
    """This function shuffles the given iterable by converting it into a
string
    and then shuffling by swapping it with a random index, so that each
possible
    order occurs with equal probability.

    Args:
        data (iter): any kind of iterable which needs to be shuffled

    Returns:
        list: list which consists of the shuffled elements
    """
    seqlen: int = len(data)
    newlist: list = list(data)
    for idx in range(seqlen):
        ridx: int = random.randint(idx, seqlen - 1)
        newlist[idx], newlist[ridx] = newlist[ridx], newlist[idx]
    return newlist


# end of shuffleseq() func.

def test_shuffleseq(seq: iter) -> None:
    """This function checks whether the given sequence is shuffled
maintaining
    its elements and its length.

    Args:
```

```
        seq (iter): any iterable sequence which needs to be shuffled and
tested for.
    """
    shuffled = shuffleseq(seq)
    print("Original:", seq)
    print("Shuffled:", shuffled)
    if set(shuffled) == set(seq) and len(shuffled) == len(seq) and shuffled
!= seq:
        print("Testcase passed")
    else:
        print("Testcase failed")


# end of test_shuffleseq() func.


# test code
if __name__ == "__main__":
    """
    Following code will be executed only when this Python
    file is run directly.  Code will be ignored if this
    file is imported by another Python source
    """

    # Testing by providing different sequences manually
    test_shuffleseq([1, 2, 3, 4, 5, 4, 3, 2, 1])
    test_shuffleseq("encryption")
    test_shuffleseq((1.2, 2.4, 3.6, 4.8, 6.0))
    test_shuffleseq({"apple", "banana", "cherry", "dates", "kiwi"})
```

Output:

```
Original: [1, 2, 3, 4, 5, 4, 3, 2, 1]
Shuffled: [3, 1, 4, 1, 4, 3, 2, 5, 2]
Testcase passed
Original: encryption
Shuffled: ['c', 'o', 'y', 'i', 'n', 't', 'n', 'e', 'r', 'p']
Testcase passed
Original: (1.2, 2.4, 3.6, 4.8, 6.0)
Shuffled: [6.0, 2.4, 3.6, 1.2, 4.8]
Testcase passed
Original: {'kiwi', 'dates', 'cherry', 'banana', 'apple'}
Shuffled: ['apple', 'cherry', 'banana', 'kiwi', 'dates']
Testcase passed
```

Explanation:

This is a Python program that provides a function shuffleseq() to shuffle any iterable in such a way that each possible order occurs with equal probability. It uses the random module to generate random indices for swapping the elements of the iterable. The program also provides a testing function test_shuffleseq() to verify if the sequence is shuffled correctly while maintaining its length and elements. If the test is passed, the program will print

"Testcase passed", otherwise it will print "Testcase failed". The test code section executes the test_shuffleseq() function with different types of sequences as input, to ensure that the shuffling function works correctly. Finally, the program contains author information and creation date.

5. The p-norm of a vector $v = (v_1, v_2, \cdots, v_n)$ 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.

Coding:

```
"""
    This module provides with a function called norm()
    which returns the p-norm form of a vector in 3d space.

    When the value of p is specifically 2, then the p-norm results
    in Euclidean norm.

    This function calculates both p-norm and Euclidean norm

    p-norm is programmatically expressed as:

    p_norm = ((v[0] ** p) + (v[1] ** p) + (v[2] ** p) + ..... + (v[n] **
p)) ** (1/p)

    where,
    v - vector in 3d space (v1, v2, v3, ...., vn)
    p - if p is 2, then results in Euclidean norm

    Original Author : Pranesh Kumar

    Created On Sat 08 Apr 2023
"""


def norm(vector: iter, p: int = 2) -> float:
    """This function calculates and returns the p-norm form of a vector.
    If the value of p is specifically 2, then it is called Euclidean form.

    When norm(vector) is called, then it calculates Euclidean form,
    where p takes the value 2, by default.
    When norm(vector, p) is called, then it calculates p-norm form.

    Args:
        vector (iter): any indexed iterable, i.e, a vector in 3d space
        p (int, optional): value of p to calculate p-norm form. Defaults to
2, for Euclidean form.

    Returns:
```

```
        float: the value of p-norm form of the given vector (sequence)
    """
    total: int = 0
    for elt in vector:
        total = total + (elt ** p)
    p_norm: float = total ** (1 / p)
    return p_norm


# end of norm() func.


# test code
if __name__ == "__main__":
    """Following code will be executed only when this Python
    file is run directly.  Code will be ignored if this
    file is imported by another Python source
    """

    # testing by providing values after getting input from user
    seq: tuple = eval(input("Enter sequence in tuple form: "))
    ch: str = input("Do you want Eucledian form? (Y?N): ")
    if ch.upper() == "Y":
        print(norm(vector=seq))
    else:
        P: int = int(input("Enter p value: "))
        print(norm(vector=seq, p=P))
```

Explanation:

This program provides a function called "norm()" that calculates and returns the p-norm form of a vector in 3D space. If the value of p is specifically 2, it returns the Euclidean norm. The function takes two arguments - a vector and the value of p (default is 2). The norm() function calculates p_norm as ((v[0] ** p) + (v[1] ** p) + ... + (v[n] ** p)) ** (1/p), where v is the input vector. The program also contains a test code that prompts the user to input a sequence in tuple form, and based on the user's input, it either returns the Euclidean form or the p-norm form of the input sequence.

==========