IP Midsem Rubric

The Best 8 out of 10 has to be considered.

Q1.
Find the output of the following python code:

```
def func1(s):
    return s[::-1]

List = ['Python','Java','C++','HTML']

for i in range(len(List)):
    List[i] = func1(List[i])

for i in range(2,len(List)):
    print(str(i-1)+". "+List[i])
```

5 marks for each print statement so 5*2 = 10 total marks

Solution:

1. ++C
2. LMTH

If a student has not added '**1.**' and '**2.**' respectively, in the answers, 1 mark + 1 mark would be deducted.

Q2. Carefully read the following codes and complete them.

a) # Program to calculate the sum of all the elements of a list
List = [2, 4, 1, 56, 1, 3, 5]
sum = 0
for i in range(len(List)):
sum += List[i]
print(sum)

b) # onlyLower function returns a copy of a string with all the lower case letters of original string
def onlyLower(s):
lowerStr = “”

```
        for ch in s:
            if ch.islower():
                lowerStr += ch
        return lowerStr
```

c)
```
# Assign a value to 's' so that "Yes" is printed out
s = _____
i = 0
j = len(s) - 1
check = 0
while i < j :
    if s[i] != s[j]:
        check = 1
    i +=1
    j -=1
if len(s) > 0 and len(s) % 5 == 0 and check == 0:
    print("Yes")
```

Length of s should be >0 and a multiple of 5. Moreover, n^{th} letter from start must be equal to n^{th} letter from end.

Some students might have given a list or a tuple instead of a string. If the list/tuple elements satisfy the constraints, marks should be given.

Marks Distribution:

a) 1.5 + 1.5
b) 1.5 + 1.5
c) 4

Q3 :

Write function **findDistinct()** in python to find the first index of each distinct element in the given list. The function findDistinct() takes a list of integers as a parameter in it and you must print all the distinct elements present in the list along with their first index (Assume 0-based indexing).

For example,

Suppose the list given as parameter is [1,2,2,1,3,4,3]

The output should be :

1:0
2:1
3:4
4:5

Explanation :

The first distinct element is 1. Its first index is 0

The next distinct element is 2. Its first index is 1.
The next distinct element is 3. Its first index is 4
The last distinct element is 4. Its first index is 5.

Note : The order of distinct elements in output can be different.
Solution:
There can be multiple valid solutions for this. The grading will work so -

1. 5 marks if the code is able to find the distinct elements
2. 5 marks if the code is able to find the correct first occurrence of an element

Q4 Find output of following code. **(3 mins)**

```
lst = [1,"banana",12,"MIDSEM"]
print(lst[1:3])
lst[2] = "IIITD"
print(lst[-2:-1])
lst[3] += "_Exam"
print(lst[3][::-1][3:8])
print(len(lst))
lst[1].replace('a','z')
lst[1] = lst[1].replace('n','q')
lst[1] = lst[1].split("q")
print(lst[1])
print(len(lst))
```

Solution:
['banana', 12]
['IIITD']
E_MES
4
['ba', 'a', 'a']
4

Marks Distribution:
['banana', 12] - 1 marks
['IIITD'] - 1 marks
E_MES - 2 mark
4 - 2 mark

['ba', 'a', 'a'] - 2 mark
4 - 2 mark

Q5:
Given a string of alphabets as parameters, mostFrequentElement() **finds the most frequent alphabet in the string and returns it. In case more than one frequent element is present, it returns any one of them.**
You are required to fix the logical errors in the code if any. (Assume syntax is correct)

```
def mostFrequentElement(str):
        list=[x for x in str]
        dictionary={}
        for i in range(1,len(list)):
                element=list[i]
                if element  not in dictionary:
                        dictionary[element] =dictionary[element]+ 1
                else:
                        dictionary[element] = 1

        mostFrequentCount= 0
        mostFrequentElement= ""
        for key in dictionary.keys():
                if mostFrequentCount > dictionary[key] :
                        mostFrequentCount=dictionary[key]
                        mostFrequentElement=dictionary[key]

        return mostFrequentElement
```

Solution: (Give 2.5 marks for each correct correction)

```
def mostFrequentElement(str):
        list=[x for x in str]
        dictionary={}
        for i in range(0,len(list)): or for i in range(1,len(list)):
                element=list[i]
                if element  not in dictionary:
                        dictionary[element] =dictionary[element]+ 1
                else:
                        dictionary[element] = 1

        mostFrequentCount= 0
        mostFrequentElement= ""
```

```
    for key in dictionary.keys():
        if mostFrequentCount < dictionary[key] :
            mostFrequentCount=dictionary[key]
            mostFrequentElement=key

    return mostFrequentElement
```

Q6

What will be the output of this program?

```
def func1():

    x = 0
    p = dict()

    while(x<=6):
        c = func2(x)
        p[x] = c
        x = x+1

    return p

def func2(B):

    t = B**3
    return t

A = func1()

print(A)
```

Solution:

```
{0: 0, 1: 1, 2: 8, 3: 27, 4: 64, 5: 125, 6: 216}
```

- func2(B) returns cube of a number.
- func1() creates an empty dictionary **p**. x→ initialized to 0. Using x as the condition check, the program iterates from x=0 to x=6. For every iteration, the value of x is stored as key of '**p**' and the value is stored in a variable "**c**" assigned by func2(x).

Marks Distribution

1. If all the values are correctly written, 10 marks (any space between elements not to be considered for evaluation)
2. Else if, Either the first element or the last element has not been printed, 8 marks
3. Else if, Both the first and the last element missing, 6 marks,

4. Else, 0 marks

Q7

From Assignment 1, we learnt to write the function findSquareDigitSum(n). This time you are required to write a function **findCubeDigitSum(n)** which returns the sum of cubes of digits of n.
For example,
For n = 231
The function should return 36 which is (2^3 + 3^3 + 1^3)

"""""""""""""""""""""""""""""""

Possible Solution:

```
def findCubeDigitSum(n):
    s = 0
    while n>0:
        digit = n%10
        s = s + (digit**3)
        n = n//10
    return s
```

Marks Distribution

1. The Function is defined with parameter '**n**' → 4 marks. If no parameter is provided, 3 marks.
2. Correct logical implementation → 4 marks [Indentation should be ignored to some extent].
3. Code should return or print, 17 on n= 122. → 2 marks.

Q8

You are given a decimal integer n and an integer k. Your task is to write a program that converts n to a number with base k.

Constraints:
1<k<10
1<n<1000

Input Format:
First-line contains the decimal integer n.
Second-line contains the base k.

Output Format:
The only line contains the number with base k.

Example Testcase:

Input :
18
4

Output:
102

Explanation:
18 = 1*(4^2) + 0*(4^1) + 2*(4^0)

Solution:

```
number = int(input())

radix = int(input())
## typecasted
converted = ''
## created an empty string

while (number > 0):
    converted = converted + str(number % radix)
    number = number // radix

reversed = converted[::-1]
print(reversed)
```

Marks Distribution

1. 2 Input statements → 2 marks (1 + 1)

2. Correct and logically similar code → 4 marks
3. For n=5 and k =2, the output should be '101' → 2 marks
4. For the sample case in the program, correct output → 2 marks

Q9

Write a code that reads a file "Scores.txt" having line-separated marks of the student and computes the total marks of the student. Finally, write the total marks in a new file called "TotalScore.txt".

"""""""""""""""""""""""""

Solution

```
f = open("Scores.txt", "r")
marks = f.readlines()
total_marks = 0
for i in marks:
    total_marks = total_marks + int(i)
f.close()

f = open("TotalScore.txt", "w")
f.write(str(total_marks))
f.close()
```

Marks distribution:

1. File 'Scores.txt' opened and read correctly → 4 Marks
2. Calculation of total marks → 2 marks
3. File 'TotalScore.txt' opened and, written correctly → 4 Marks.

Q10

You are provided with two dictionaries named "**marks_obtained**" and "**total marks**". These dictionaries contain marks obtained in a particular subject and total marks allocated for that subject. The **keys** to this dictionary are the subjects themselves.

```
marks_obtained = {"IP":85, "PSY":93, "DC":24, "LA": 32}
total_marks    = {"IP":100, "PSY":100, "DC":40, "LA": 40}
```

a)Your task is to write a program that will return a dictionary of the percentage of marks scored in each of the subjects.

Remember → keys will be the same for both the dictionaries. The subjects can be different other than the ones mentioned in the snippet above.

The output for the aforementioned dictionaries, itself will be a dictionary as follows:

```
{'IP': 85.0, 'PSY': 93.0, 'DC': 60.0, 'LA': 80.0}
```

b) Would there be any problem if the dictionaries are:

```
marks_obtained = {"IP":85, "PSY":93, "DC":24, "LA": 32}
total_marks    = {"IP":100, "PSY":100, "LA": 40, "DC":40}
```

Why/ Why not? Justify in a line.

Possible Solution:

a)

```
##
marks_obtained = {"IP":85, "PSY":93, "DC":24, "LA": 32}
total_marks    = {"IP":100, "PSY":100, "DC": 40, "LA":40}

b = {} ## or using dict()
## created empty dictionary

for k in marks_obtained.keys():
  a = (marks_obtained[k]/total_marks[k])*100
  b[k]=a
##loop ends
print(b)
```

b)

No. Order of the keys does not matter unless it has explicitly been mentioned, that the said dictionary maintains order (using OrderedDict or some other paradigm)

<u>Marks Distribution:</u>

a)

1. Correct implementation(logically similar to the solution, can vary from program to program) → 2 marks.
2. All keys from marks_obtained, are present in the final output, with correct % values → 4 marks.

b)

1. No → 2 marks
2. Justification → hinting towards “order” and similar to some extent to the correct explanation → 2 marks.

In all of the questions, if the code is logically correct, indentation and spacing would be ignored and will not be considered for the evaluation.