Object-Oriented Design Lab Report (Python 2)

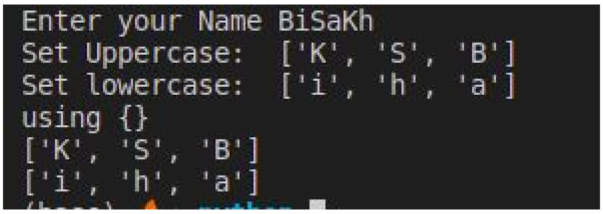
Bisakh Mondal 001810501079

Format: Question | Approach(if any) | Code | Output

Python Assignment

(Q1) Compare uppercase and lowercase of your name using set() and {} syntax.
Set in python is an example of an unordered, unindexed container with unique elements. The set difference will do so.

```
name= input('Enter your Name ')

uppercase= list(set(name)-set(name.lower()))
lowercase= list(set(name)-set(name.upper()))

print('Set Uppercase: ',uppercase)
print('Set lowercase: ',lowercase)

#using {}

uppercase1 = list( {*name}.difference({*name.lower()}))
lowercase1 = list( {*name}.difference({*name.upper()}))

print('using {}')
print(uppercase1)
print(lowercase1)
```



```
Enter your Name BiSaKh
Set Uppercase:  ['K', 'S', 'B']
Set lowercase:  ['i', 'h', 'a']
using {}
['K', 'S', 'B']
['i', 'h', 'a']
```

(Q2) Write first seven Fibinacci numbers using generator next function/ yield in python.

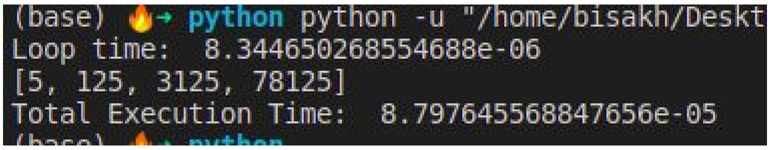
Fibonacci sequence generator using yield.

```
def FibGen():
    a,b,cnt =0,1,7
    while cnt:
        yield a
        a,b = b, a+b
        cnt-=1
    # raise StopIteration()

lis = list(FibGen())
print(lis)
```

(**Q3**)Write a code which yields all terms of the geometric progression a, aq, aq 2 , aq 3 , ... When the progression produces a term that is greater than 100,000, the generator stops (with a return statement). Compute total time and time within the loop.

Generator with return statement raises a **StopIteration** with error value the same as the return value. If the sequence value exceeds 100,000 then it stops.

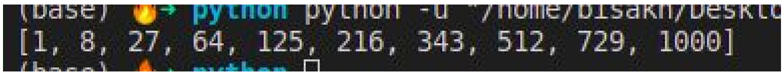
```
import time

start = time.time()

def geometricProgression(a,q):
    while True:
        if a> 100000:
            print("Loop time: ", time.time()- start)
            return "Iteration is Done" #Like
StopIteration(message)
        yield a
        a=a*q


x= geometricProgression(5,25)
lis= list(x)

print(lis)
print("Total Execution Time: ", time.time()-start)
# x.__next__()
```



```
(base) 🔥→ python python -u "/home/bisakh/Deskt
Loop time:  8.344650268554688e-06
[5, 125, 3125, 78125]
Total Execution Time:  8.797645568847656e-05
(base) 🔥→ python
```

(**Q4**) Create a generator expression for first 10 cubes.

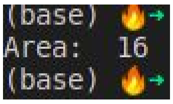
```
def CubeGen():
    cnt=1
    while cnt<=10:
        yield cnt*cnt*cnt
        cnt+=1

print(list(CubeGen()))
```



```
(base) 🔥→ python python -u "/home/bisakh/Deskto
[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
(base) 🔥→ python □
```

(**Q5**) Write a program to compute square area of square class with self() to get square value in python.

```
class square:
    def __init__(self, len=0):
        self.len=len

    def self(self):
        return self.len*self.len

s= square(4)
print("area: ",s.self())
```

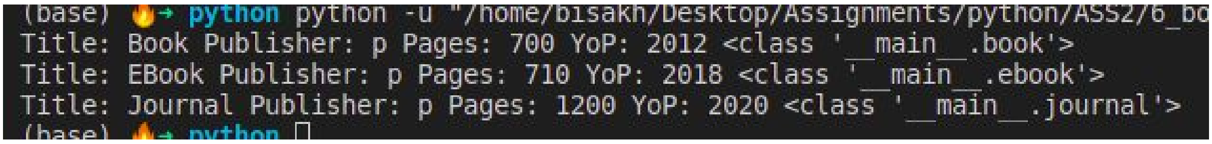


```
(base) 🔥→
Area:  16
(base) 🔥→
```

(**Q6**) Create book, ebook, journal classes to use inheritance with title, publisher, page, year of publishing details.

```
class book:
    def __init__(self, title, pub, page, year):
        self.title= title
        self.pub = pub
        self.page=page
        self.yearOfPublicaton=year
    def __str__(self):
        return 'Title: {} Publisher: {} Pages: {} YoP:
{}'.format(self.title, self.pub,self.page,self.yearOfPublicaton)

class ebook(book):
    pass
```

```
class journal(book):
    pass

b=book("Book",'p',700,2012)
print(b, type(b))
EB=ebook("EBook",'p',710,2018)
print(EB, type(EB))
J=journal("Journal",'p',1200,2020)
print(J, type(J))
```

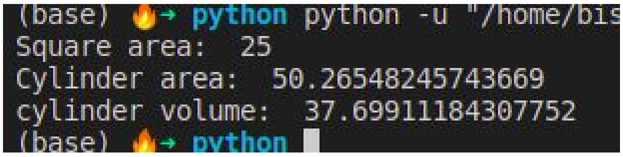


```
(base) 🔥→ python python -u "/home/bisakh/Desktop/Assignments/python/ASS2/6_bo
Title: Book Publisher: p Pages: 700 YoP: 2012 <class '__main__.book'>
Title: EBook Publisher: p Pages: 710 YoP: 2018 <class '__main__.ebook'>
Title: Journal Publisher: p Pages: 1200 YoP: 2020 <class '__main__.journal'>
(base) 🔥→ python □
```

(**Q7**)Show multiple inheritance in shape, 2-D shapes, 3-D shapes, square, rectangle, polygon, hexagon, cube, cone, cylinder etc. classes with their areas.

Approach: it’s wise to use **abstract class and abstract methods** for the base classes like Shape2D and Shape3D and implement those specific abstract methods like **area**, **volume** in the inherited classes. That way we can prevent objects from being initiated from those abstract classes directly.

```
from abc import ABC, abstractmethod
import math

class shape2D(ABC):
    @abstractmethod
    def area(self):
        pass

class shape3D(ABC):
    @abstractmethod
    def volume(self):
        pass

class square(shape2D):

    @classmethod
    def area(cls, len):
        return len*len

class rectangle(shape2D):
    @classmethod
    def area(cls, lena, lenb):
        return lena*lenb

class hexagon(shape2D):
    @classmethod
    def area(cls, len):
        return 3*math.sqrt(3)*len*len/2

class cube(shape2D, shape3D):
    @classmethod
    def area(cls, len):
        return 6*len*len

    @classmethod
    def volume(cls,len):
        return len*len*len

class cone(shape2D, shape3D):
    @classmethod
    def area(cls, radius, height):
        return math.pi*math.pow(radius,2) + \

math.pi*radius*math.sqrt(math.pow(radius,2)+math.pow(height,2))

    @classmethod
    def volume(cls,radius,height):
        return math.pi*math.pow(radius,2)*height/3

class cylinder(shape2D, shape3D):
    @classmethod
    def area(cls, radius, height):
        return math.pi*math.pow(radius,2) + \
            2*math.pi*radius*height

    @classmethod
    def volume(cls,radius,height):
        return math.pi*math.pow(radius,2)*height


print(square.area(5))
print(cylinder.area(2,3))
print(cylinder.volume(2,3))
```



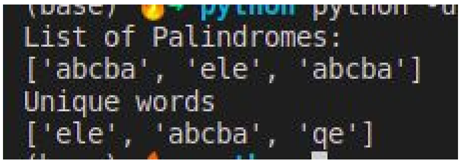
```
(base) 🔥→ python python -u "/home/bis
Square area:  25
Cylinder area:  50.26548245743669
cylinder volume:  37.69911184307752
(base) 🔥→ python
```

(**Q8**) Search for palindrome and unique words in a text using class method and string method.

Splitting the sentence into tokens and filtering the list using a Lamba function and string slicing palindromic words can be checked.

And for unique words the whole word list can be cast to set and then formed list again.

```
class StringManipulate:
    @classmethod
    def palindromes(cls, st):
        wordList= st.split()

        filterPal = lambda word : word == word[::-1]

        palins = list(filter(filterPal, wordList))
        print("List of Palindromes: ")
        print(palins)

        return palins

    @classmethod
    def UniqueWords(cls, st):
        uni = list(set(st.split()))
        return uni

st= "abcba ele abcba qe"

StringManipulate.palindromes(st)
print("Unique words")
print(StringManipulate.UniqueWords(st))
```

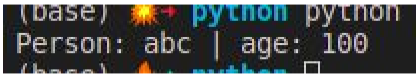


```
List of Palindromes:
['abcba', 'ele', 'abcba']
Unique words
['ele', 'abcba', 'qe']
```

(**Q9**) Check and set a person's age in person class using property decorator.

Decorators in python allow specific functions to be wrapped by another functions or class.
The property decorator in python allow get, set, delete operation on data members in pythonic way.

```
class Person:
    def __init__(self,name='',age=0):
        self.__name=name
        self.__age=age
    @property
    def age(self):
        return self.__age
    @property
    def name(self):
        return self.__name

    @age.setter
    def age(self,age):
        self.__age=age

    @name.setter
    def name(self,name):
        self.__name=name

    def __str__(self):
        return f'Person: {self.name} | age: {self.__age}'

p=Person('Bisakh')

p.age=10
p.name='abc'
print(p)
```

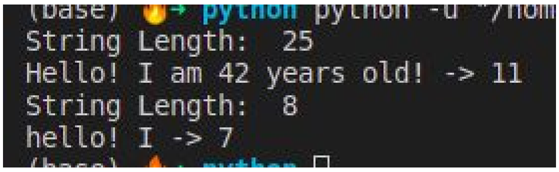


```
(base)  python python
Person: abc | age: 100
```

(**Q10**) Write a operator overloading for “len” which shows string length for any given string and return only length of last three words if the string is in "Hello! I am 42 years old!" format.

In **NewString class** , the **__len__** magic method is overloaded with a return of splitting the sentence and counting length of last three words.

```
class NewString:
    def __init__(self, st=''):
        self.str= st
        if not isinstance(self.str, str):
            raise TypeError("Expected arguement type string")

    def __len__(self):
        wordLis= self.str.split()
        print("String Length: ",len(self.str))
        length=0
        for w in range(max(0,len(wordLis)-3),len(wordLis)):
            length+=len(wordLis[w])
        return length

s= NewString("Hello! I am 42 years old!")
print(len(s))
s= NewString("Hello! I")
print(len(s))
```

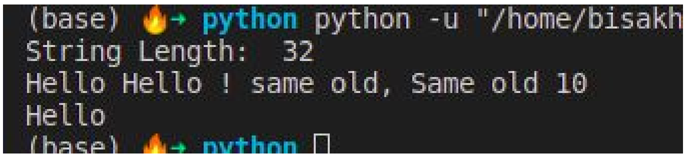


```
(base) 🔥→ python python -u ~/hom
String Length:  25
Hello! I am 42 years old! -> 11
String Length:  8
hello! I -> 7
(base) 🔥→ python □
```

(**Q11**) Write a operator overloading for “len” which shows string length for any given string and return only length of repetitive words with the text if the text has some repetitive parts. Determine the most frequently occurring words using most_common.

len is returning the repetitive words length by keeping a count on dictionary and the most common words are calculated using the maxfunction and the count member function of the list.

```
class NewString:
    def __init__(self,st) -> None:
        self.str=st

    def __len__(self):
        print("String Length: ",len(self.str))
        wordList = self.str.split()
        mapW = {}
        length=0
        for w in wordList:
            mapW[w] = mapW.get(w,0)+1

        for k,v in mapW.items():
            if v>1:
                length+=len(k)*v
        return length
    def most_common(self):
        wordlist= self.str.split()
        return max(wordlist, key=wordlist.count)

    def __str__(self) -> str:
        return self.str
s= NewString("Hello Hello ! same old, Same old")

print(s,len(s))
print(s.most_common())
```



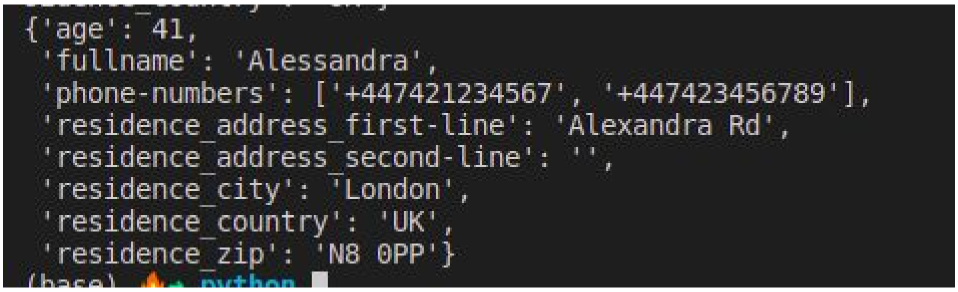
```
(base) 🔥→ python python -u "/home/bisakh
String Length:  32
Hello Hello ! same old, Same old 10
Hello
(base) 🔥→ python ▯
```

(**Q12**) Write a function that flattens a nested dictionary structure like one obtained from Twitter and Facebook APIs or from some JSON file.

Approach: The problem is a very good example of a graph traversing algorithm like **DFS or BFS** and adding parents to the front of children the whole JSON object can be flattened.

```
import collections

nested = {
'fullname': 'Alessandra',
'age': 41,
'phone-numbers': ['+447421234567', '+447423456789'],
'residence': {
'address': {
'first-line': 'Alexandra Rd',
'second-line': '',
    },
'zip': 'N8 0PP',
'city': 'London',
'country': 'UK',
},
}

seperator='_'
def flatten(dictionary,parent=''):
    tempList=[]
    for key, val in dictionary.items():
        Nkey = key if parent=='' else parent+seperator+key
        if isinstance(val, collections.MutableMapping):
            tempList.extend(flatten(val, Nkey).items())
        else:
            tempList.append((Nkey,val))
    return dict(tempList)
```

```
flattend= flatten(nested)

print(flattend)
```

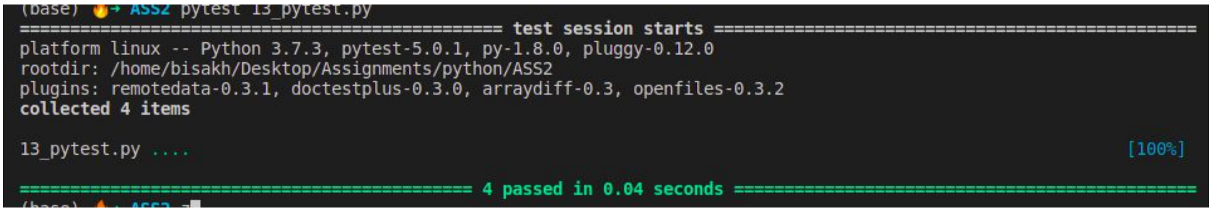


```
{'age': 41,
 'fullname': 'Alessandra',
 'phone-numbers': ['+447421234567', '+447423456789'],
 'residence_address_first-line': 'Alexandra Rd',
 'residence_address_second-line': '',
 'residence_city': 'London',
 'residence_country': 'UK',
 'residence_zip': 'N8 0PP'}
(base) 🔥→ python █
```

(**Q13**)
Use parameterized or nose_parameterized to compute power of following values:
(2, 2, 4),
(2, 3, 8),
(1, 9, 1),
(0, 9, 0). Use pytest to check errors.

Parameterized tests are generally created to create multiple tests of same type without writing the same code.

```
import pytest

def power(a,b):
    return a**b

@pytest.mark.parametrize('a,b,c',[[2,2,4],[2,3,8],[1,9,1],[0,9,0]
])
def test(a,b,c):
    print(type(a))
    assert power(a,b) == c
```



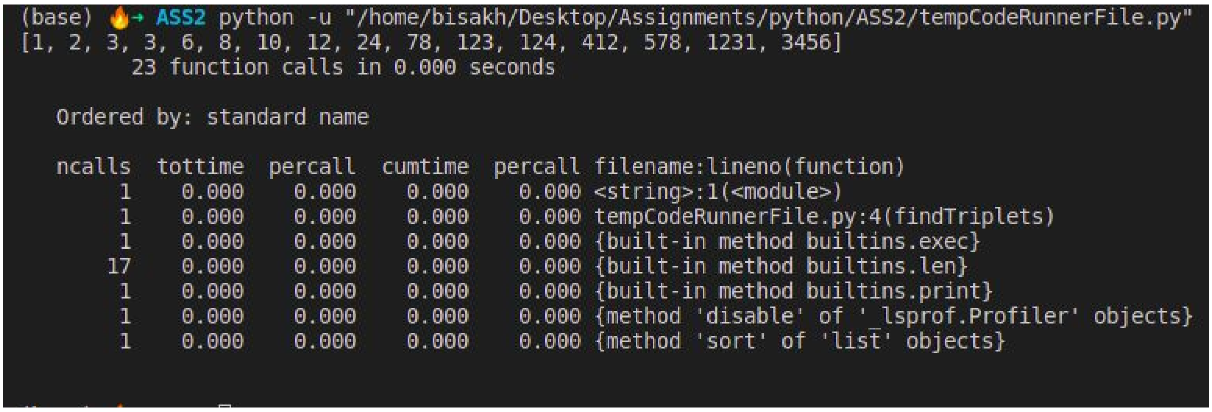
```
(base) 🔥→ ASS2 pytest 13_pytest.py
=============================================== test session starts ================================================
platform linux -- Python 3.7.3, pytest-5.0.1, py-1.8.0, pluggy-0.12.0
rootdir: /home/bisakh/Desktop/Assignments/python/ASS2
plugins: remotedata-0.3.1, doctestplus-0.3.0, arraydiff-0.3, openfiles-0.3.2
collected 4 items

13_pytest.py ....                                                                                            [100%]

============================================= 4 passed in 0.04 seconds =============================================
(base) 🔥→ ASS2 █
```

(**Q14**) Use profile/cprofile to check pythogorian triples code in python. Think about time complexity of the code.

cProfile in python provides a deterministic profiling i.e statistics regarding how long, often the parts of a program run, etc.

For the time complexity, the program runs in O(N*N) where N is the number of inputs(array size)

```
import cProfile

#Complexity O(N*N)
def findTriplets(ar):
    ar.sort()
    print(ar)
    for i in range(len(ar)):
        j, k=i+1, len(ar)-1

        while j<=k:
            if ar[j]*ar[j] + ar[i]*ar[i]==ar[k]*ar[k]:
                print("Triplets: ",ar[i],ar[j],ar[k])
                return True
            if ar[j]*ar[j] + ar[i]*ar[i] > ar[k]*ar[k]:
                k-=1
            else:
                j+=1
    return False


arr = [3,2,1,8,6,10,12,24,1231,123,124,3456,578,78,412,3]

cProfile.run('findTriplets(arr)')
```

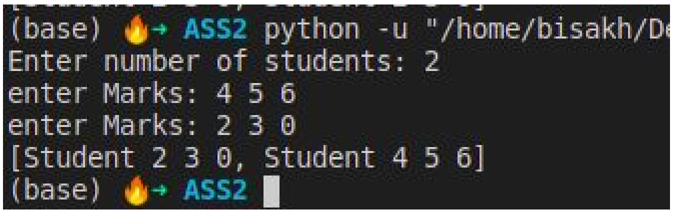


```
(base) 🔥➜ ASS2 python -u "/home/bisakh/Desktop/Assignments/python/ASS2/tempCodeRunnerFile.py"
[1, 2, 3, 3, 6, 8, 10, 12, 24, 78, 123, 124, 412, 578, 1231, 3456]
         23 function calls in 0.000 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.000    0.000 <string>:1(<module>)
        1    0.000    0.000    0.000    0.000 tempCodeRunnerFile.py:4(findTriplets)
        1    0.000    0.000    0.000    0.000 {built-in method builtins.exec}
       17    0.000    0.000    0.000    0.000 {built-in method builtins.len}
        1    0.000    0.000    0.000    0.000 {built-in method builtins.print}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
        1    0.000    0.000    0.000    0.000 {method 'sort' of 'list' objects}
```

(**Q15**)Write a program to sort in descending order by the sum of credits accumulated by students, so to have the best student at position 0. Write a function using map, to produce a decorated object, to sort, and then to undecorate. Each student has credits in three (possibly different) subjects. To decorate an object means to transform it, either adding extra data to it, or putting it into another object, in a way that allows to sort the original objects the way you want. After the sorting, one reverts the decorated objects to get the original ones from them. This is called to undecorate.

Just creating a list of tuples by augmenting each student with their corresponding ,marks then after sorting based on tuple second element undecorate it.

```
class Student:
    def __init__(self,m1=0,m2=0,m3=0) -> None:
        self.m1=m1
        self.m2=m2
        self.m3=m3

    def sum(self):
        return self.m1+self.m2+self.m3

    def __repr__(self) -> str:
        return f"Student {self.m1} {self.m2} {self.m3}"
sList=[]
for _ in range(int(input("Enter number of students: "))):
    marks = input("enter Marks: ")
    m1, m2, m3 = [int(x) for x in marks.split()]
    sList.append(Student(m1,m2,m3))

decorated = [(stud, stud.sum()) for stud in sList]

decorated.sort(key=lambda x: x[1])

undec= [x[0] for x in decorated]
print(undec)
```

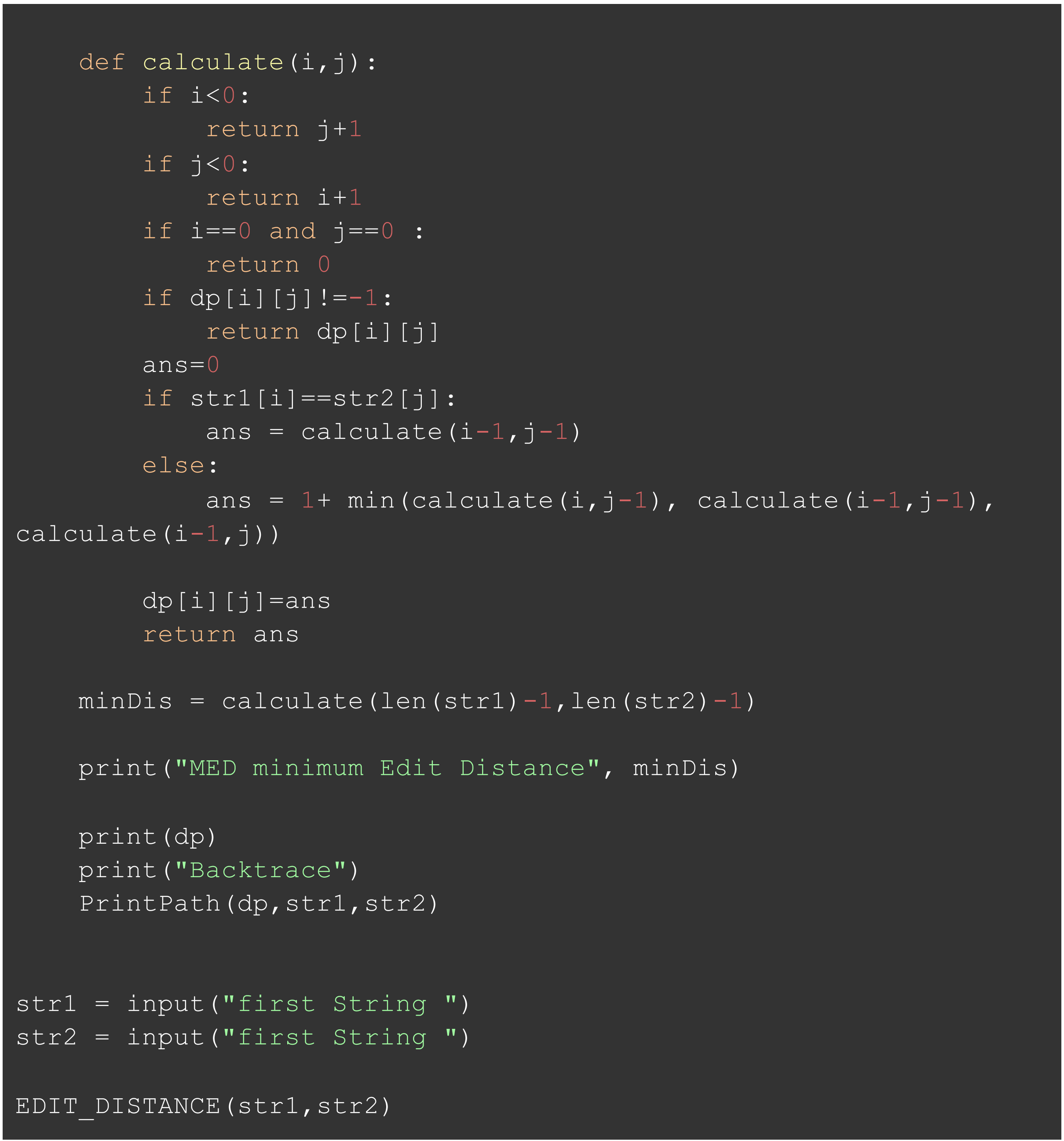


```
(base) 🔥→ ASS2 python -u "/home/bisakh/De
Enter number of students: 2
enter Marks: 4 5 6
enter Marks: 2 3 0
[Student 2 3 0, Student 4 5 6]
(base) 🔥→ ASS2 █
```

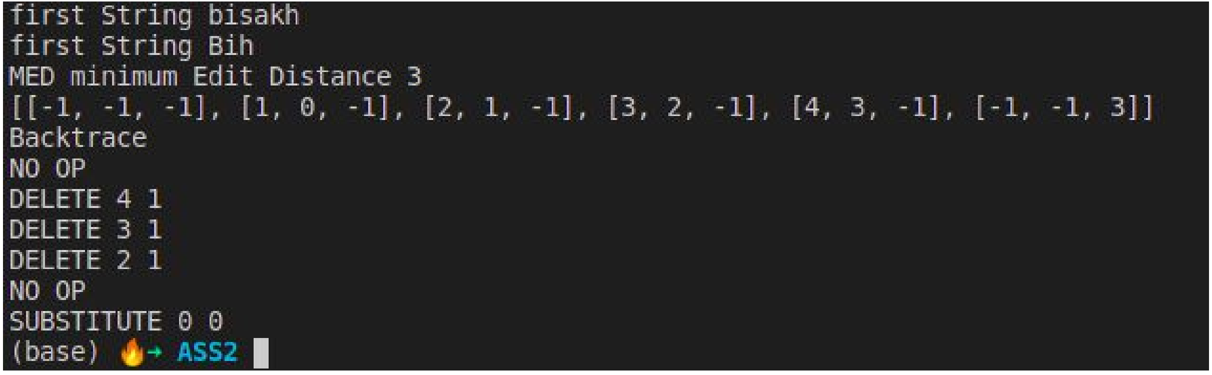
(**Q16**) Write a python program to calculate the number of editing operations (substitution, deletion and insertion) in the output sequence with respect to a given reference input. Prepare the Minimum Edit Distance (MED) Table and print the backtrace to MED (Consider the root form of words while calculating the number of editing operations)

It’s Classical **Dynamic programming** problem named as EditDistance, as along with providing the MED the backtrace has to be provided, So I liked to used a **Closure** and print the backtrace from the MED table.

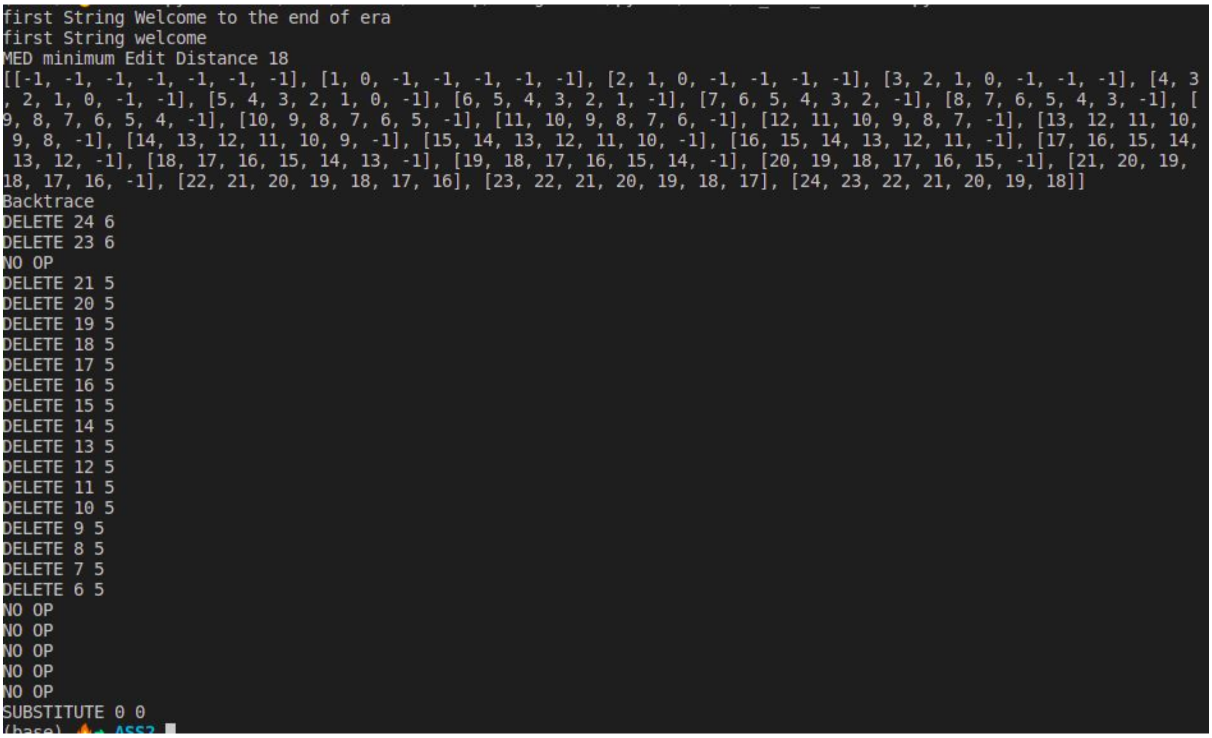
```
def PrintPath(dp,str1,str2):
    i,j = len(str1)-1, len(str2)-1

    while i >=0 and j>=0:
        if str1[i]==str2[j]:
            print("NO OP")
            i-=1
            j-=1
            if i<0 :
                while j>=0:
                    print("DELETE")
                    j-=1

            if j<0 :
                while i>=0:
                    print("INSERT")
                    i-=1
        elif i>0 and dp[i][j]==dp[i-1][j]+1:
            print("DELETE", i,j)
            i-=1
        elif j>0 and dp[i][j] == dp[i][j-1]+1:
            print("INSERT",i,j)
            j-=1
        elif i>0 and j>0 and dp[i][j] == dp[i-1][j-1]+1:
            print("SUBSTITUTE",i,j)
            i-=1
            j-=1
        else:
            print("SUBSTITUTE",i,j)
            i-=1
            j-=1


def EDIT_DISTANCE(str1, str2):
    dp = [[-1]*len(str2) for _ in range(len(str1))]
```



```
    def calculate(i,j):
        if i<0:
            return j+1
        if j<0:
            return i+1
        if i==0 and j==0 :
            return 0
        if dp[i][j]!=-1:
            return dp[i][j]
        ans=0
        if str1[i]==str2[j]:
            ans = calculate(i-1,j-1)
        else:
            ans = 1+ min(calculate(i,j-1), calculate(i-1,j-1),
calculate(i-1,j))

        dp[i][j]=ans
        return ans

    minDis = calculate(len(str1)-1,len(str2)-1)

    print("MED minimum Edit Distance", minDis)

    print(dp)
    print("Backtrace")
    PrintPath(dp,str1,str2)


str1 = input("first String ")
str2 = input("first String ")

EDIT_DISTANCE(str1,str2)
```



```
first String bisakh
first String Bih
MED minimum Edit Distance 3
[[-1, -1, -1], [1, 0, -1], [2, 1, -1], [3, 2, -1], [4, 3, -1], [-1, -1, 3]]
Backtrace
NO OP
DELETE 4 1
DELETE 3 1
DELETE 2 1
NO OP
SUBSTITUTE 0 0
(base) 🔥→ ASS2 
```



```
first String Welcome to the end of era
first String welcome
MED minimum Edit Distance 18
[[-1, -1, -1, -1, -1, -1, -1], [1, 0, -1, -1, -1, -1, -1], [2, 1, 0, -1, -1, -1, -1], [3, 2, 1, 0, -1, -1, -1], [4, 3
, 2, 1, 0, -1, -1], [5, 4, 3, 2, 1, 0, -1], [6, 5, 4, 3, 2, 1, -1], [7, 6, 5, 4, 3, 2, -1], [8, 7, 6, 5, 4, 3, -1], [
9, 8, 7, 6, 5, 4, -1], [10, 9, 8, 7, 6, 5, -1], [11, 10, 9, 8, 7, 6, -1], [12, 11, 10, 9, 8, 7, -1], [13, 12, 11, 10,
 9, 8, -1], [14, 13, 12, 11, 10, 9, -1], [15, 14, 13, 12, 11, 10, -1], [16, 15, 14, 13, 12, 11, -1], [17, 16, 15, 14,
 13, 12, -1], [18, 17, 16, 15, 14, 13, -1], [19, 18, 17, 16, 15, 14, -1], [20, 19, 18, 17, 16, 15, -1], [21, 20, 19,
18, 17, 16, -1], [22, 21, 20, 19, 18, 17, 16], [23, 22, 21, 20, 19, 18, 17], [24, 23, 22, 21, 20, 19, 18]]
Backtrace
DELETE 24 6
DELETE 23 6
NO OP
DELETE 21 5
DELETE 20 5
DELETE 19 5
DELETE 18 5
DELETE 17 5
DELETE 16 5
DELETE 15 5
DELETE 14 5
DELETE 13 5
DELETE 12 5
DELETE 11 5
DELETE 10 5
DELETE 9 5
DELETE 8 5
DELETE 7 5
DELETE 6 5
NO OP
NO OP
NO OP
NO OP
NO OP
SUBSTITUTE 0 0
(base) ASS2
```

(**Q17**) Write a single python program to do the following operations on a text file by writing different user defined functions.

a. Remove all the special characters.

b. Remove all single characters.

c. Substitute multiple spaces with single space.

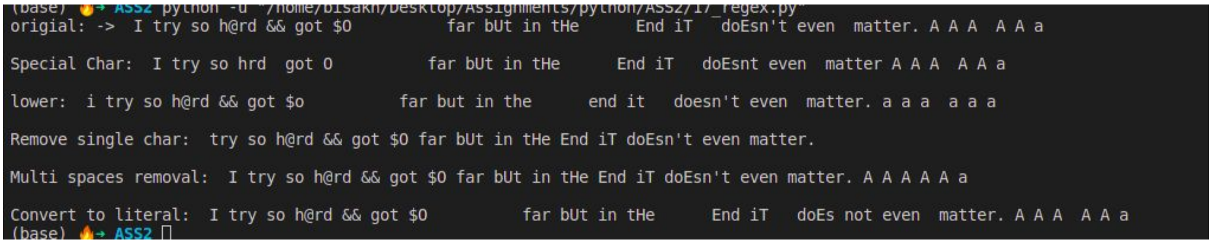
d. Convert all the words into Lowercase.

e. Convert the words into literal form from their contracted form (e.g., Couldn't Could not).

Approach: it's an ideal problem of regex. Instead of coding monotonous python, regex capabilities can be hugely exploited here.

```
import re

def RemoveSpecial(text):

    #Regex to replace all but the normal chars
    retext= re.sub('[^ a-zA-Z0-9]+','',text)
    return retext

def RemoveSingleChar(text):
    return ' '.join(i for i in text.split() if len(i)>1)

def RemoveMultipleSpaces(text):
    retext= re.sub('[ ]+',' ',text)
    return retext
```

```
def TOLOWER(text):
    return text.lower()

def convertToLiteral(text):
    text = re.sub('n\'t', " not",text)
    text = re.sub('\'ll', " will",text)
    text = re.sub('\'d', " would",text)
    text = re.sub('\'ve', " have",text)
    text = re.sub('\'s', " is",text)
    return text


t="I try so h@rd && got $O          far bUt in tHe      End iT
doEsn't even  matter. A A A  A A a"

print("origial: -> ",t)
print("Special Char: ", RemoveSpecial(t))
print("lower: ", TOLOWER(t))
print("Remove single char: ", RemoveSingleChar(t))
print("Multi spaces removal: ", RemoveMultipleSpaces(t))
print("Convert to literal: ", convertToLiteral(t))
```

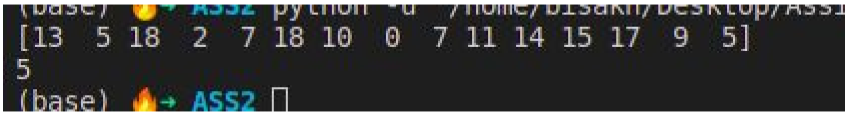


```
(base) 🔥→ ASS2 python -u "/home/bisakh/Desktop/Assignments/python/ASS2/17_regex.py"
origial: ->  I try so h@rd && got $O          far bUt in tHe      End iT   doEsn't even  matter. A A A  A A a

Special Char:  I try so hrd  got O          far bUt in tHe      End iT   doEsnt even  matter A A A  A A a

lower:  i try so h@rd && got $o          far but in the      end it   doesn't even  matter. a a a  a a a

Remove single char:  try so h@rd && got $O far bUt in tHe End iT doEsn't even matter.

Multi spaces removal:  I try so h@rd && got $O far bUt in tHe End iT doEsn't even matter. A A A A A a

Convert to literal:  I try so h@rd && got $O          far bUt in tHe      End iT   doEs not even  matter. A A A  A A a
(base) 🔥→ ASS2 ▯
```

(**Q18**) Using Numpy create random vector of size 15 having only Integers in the range 0 -20. Write a program to find the most frequent item/value in the vector list.

Approach: USing numpy’s built in api random’s randint is used to create vector of desired shape. And for most frequent item max count is used .

```
import numpy as np

rands = np.random.randint(low=0, high=20, size=15)

print(rands)
print(max(rands, key=rands.tolist().count))
```

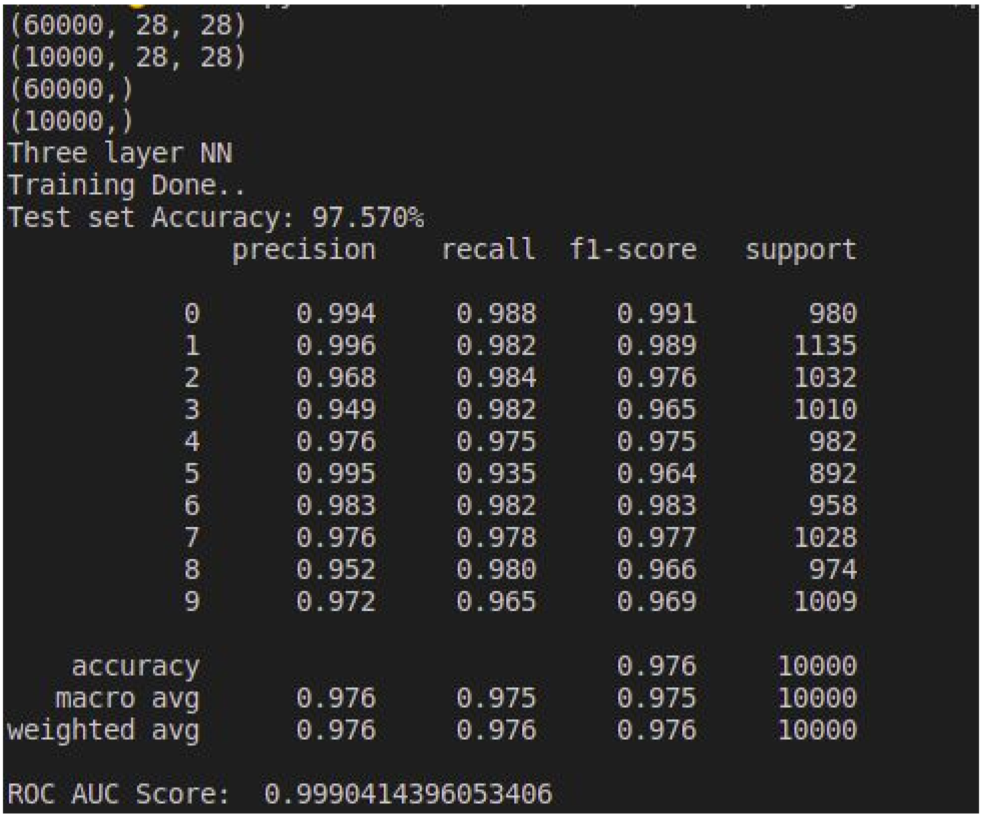


```
(base) 🔥→ ASS2 python -u "/home/bisakh/Desktop/Ass1
[13  5 18  2  7 18 10  0  7 11 14 15 17  9  5]
5
(base) 🔥→ ASS2 ▯
```

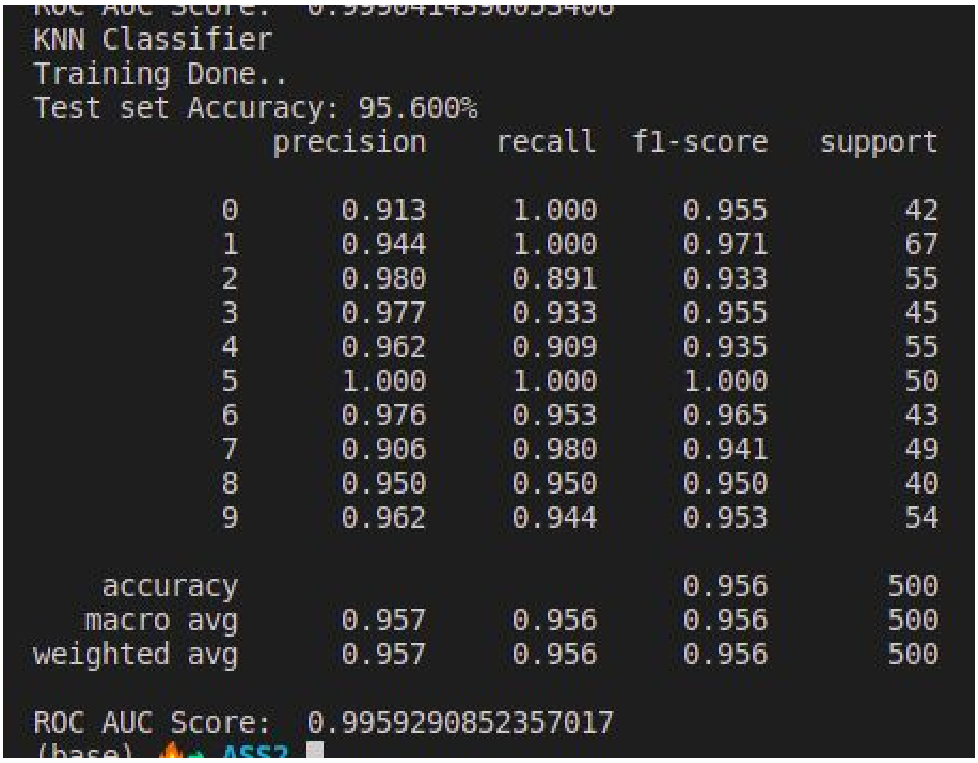
(**Q19**) **MNIST problem:**
Used K Nearest Neighbor and 3 Layer Neural net (3 hidden Layers) of shape (256,128,64)

```
import numpy as np
import gzip
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, roc_auc_score
from sklearn.neural_network import MLPClassifier

testLoc={
    'image':'MNIST_DATA/t10k-images-idx3-ubyte.gz',
    'label': 'MNIST_DATA/t10k-labels-idx1-ubyte.gz'
}
trainLoc={
    'image':'MNIST_DATA/train-images-idx3-ubyte.gz',
    'label': 'MNIST_DATA/train-labels-idx1-ubyte.gz'
}

def extract_Images(filename):
    f=gzip.open(filename,'r')
    f.read(4) #magic number
    nImgs= int.from_bytes(f.read(4),'big')
    row= int.from_bytes(f.read(4),'big')
    col= int.from_bytes(f.read(4),'big')
    data=f.read()
    images=
np.frombuffer(data,dtype=np.uint8).astype(np.float32).reshape((nI
mgs,row,col))
    return images

def extract_Labels(filename):
    file = gzip.open(filename,'rb')
    file.read(8)
    data= file.read()
    labs = np.frombuffer(data,dtype=np.uint8).astype(np.int32)
    return labs

trainImg = extract_Images(trainLoc['image'])
trainLab = extract_Labels(trainLoc['label'])

testImg = extract_Images(testLoc['image'])
testLab = extract_Labels(testLoc['label'])
```

```
print(trainImg.shape)
print(testImg.shape)
print(trainLab.shape)
print(testLab.shape)

#TRAIN function
def train(images, labels, classifier):
    images = images.reshape(images.shape[0],-1)
    classifier.fit(images,labels)

    print("Training Done..")

    ''' Training set Performance (but costly ops) '''
    # output = classifier.predict(images)
    # train_acc = (output==labels).sum()/labels.shape[0]
    # print("Training Accuracy: ", train_acc)

#TEST function
def test(images,labels,classifier):
    images = images.reshape(images.shape[0],-1)
    op = classifier.predict(images)
    # train(trainImg,trainLab,NN)

    test_acc = (op==labels).sum()/labels.shape[0]
    print("Test set Accuracy: {:.3f}%".format(test_acc*100))
    clReport= classification_report(labels,op, digits=3)
    print(clReport)
    auc = roc_auc_score(labels, classifier.predict_proba(images),
multi_class="ovr")
    print("ROC AUC Score: ",auc)



KNN = KNeighborsClassifier(n_neighbors=10)

NN = MLPClassifier(random_state=1, solver= 'adam', max_iter=300,
hidden_layer_sizes=(256,128,64))

#NN
print("Three layer NN")
train(trainImg,trainLab,NN)
test(testImg,testLab,NN)

#Training Testing in KNN classifier
print("KNN Classifier")
train(trainImg,trainLab,KNN)
test(testImg[:500],testLab[:500],KNN)
```



```
(60000, 28, 28)
(10000, 28, 28)
(60000,)
(10000,)
Three layer NN
Training Done..
Test set Accuracy: 97.570%
              precision    recall  f1-score   support

           0      0.994     0.988     0.991       980
           1      0.996     0.982     0.989      1135
           2      0.968     0.984     0.976      1032
           3      0.949     0.982     0.965      1010
           4      0.976     0.975     0.975       982
           5      0.995     0.935     0.964       892
           6      0.983     0.982     0.983       958
           7      0.976     0.978     0.977      1028
           8      0.952     0.980     0.966       974
           9      0.972     0.965     0.969      1009

    accuracy                          0.976     10000
   macro avg      0.976     0.975     0.975     10000
weighted avg      0.976     0.976     0.976     10000

ROC AUC Score:  0.9990414396053406
```



```
ROC AUC Score:  0.9990414396053406
KNN Classifier
Training Done..
Test set Accuracy: 95.600%
              precision    recall  f1-score   support

           0      0.913     1.000     0.955        42
           1      0.944     1.000     0.971        67
           2      0.980     0.891     0.933        55
           3      0.977     0.933     0.955        45
           4      0.962     0.909     0.935        55
           5      1.000     1.000     1.000        50
           6      0.976     0.953     0.965        43
           7      0.906     0.980     0.941        49
           8      0.950     0.950     0.950        40
           9      0.962     0.944     0.953        54

    accuracy                          0.956       500
   macro avg      0.957     0.956     0.956       500
weighted avg      0.957     0.956     0.956       500

ROC AUC Score:  0.9959290852357017
(base)   ASS2
```

(Q20)

Its a very good guided problem for class, inheritance polymorphism etc.

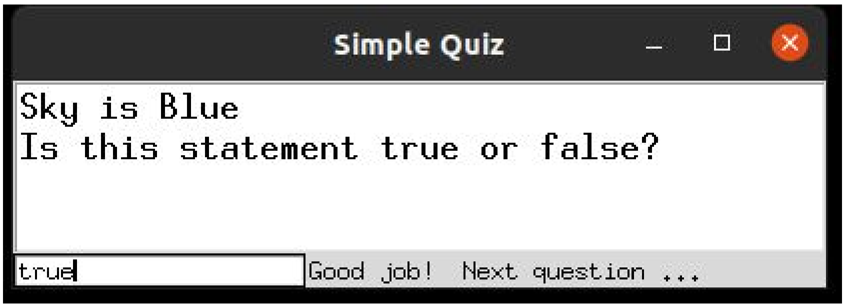
Problem.py

```
'''
Advanced OOPs

@file problem.py
@author: Bisakh Mondal
'''

class Problem:
    ''' Base Problem class '''

    def __init__(self, text):
        self.text = text

    def get_text(self):
        return self.text
```

Shortanswer.py

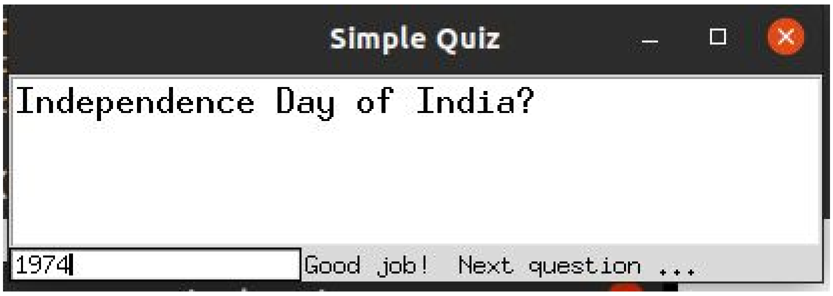
```
'''
Advanced OOPs

@author: Bisakh Mondal
'''
from problem import Problem

class ShortAnswer(Problem):
    ''' Model a short-answer question '''

    def __init__(self, q, a):
        ''' Construct a short-answer question with question and
answer texts '''
        super().__init__(q)
        self.answer = a

    def ask_question(self):
        ''' Return the question text '''
        return self.get_text() + '?'

    def check_answer(self, a):
        ''' Return True if a is the correct answer; False
otherwise '''
        return self.answer == a
    def get_answer(self):
        ''' Return the answer text '''
        return self.answer


class FillInTheBlack(ShortAnswer):
    ''' Model a Fill in the Blank question '''

    ''' Overrided Function from ShortAnswer parent with necessary
Modification'''
    def ask_question(self):
        ''' Return the question text with necessary Instruction
'''
        return self.get_text() + '\nFill in the blank.'


class TrueFalse(ShortAnswer):
    ''' Model a True False question '''

    ''' Overrided Constructor from ShortAnswer parent with answer
type checking '''
    def __init__(self, q, a):
        if not isinstance(a, bool):
            raise TypeError("Answer is expected of type bool")

        #Calling constructor of parent class with suitable
arguements
        super().__init__(q, a)

    ''' Overrided Function from ShortAnswer parent with Modified
question'''
    def ask_question(self):
        ''' Return the question text with necessary Instruction
'''
        return self.get_text() + '\nIs this statement true or
false?'

    ''' Overrided Function from ShortAnswer parent '''
    def get_answer(self):
        ''' Return the answer in string '''
        return str(self.answer)

    ''' Overrided Function from ShortAnswer parent '''
    def check_answer(self, a):
        ''' Return True if a (in title case) is the correct
answer; False otherwise '''
        return str(self.answer) == a.title()
```

```
if __name__ == "__main__":
    q = ShortAnswer('question', 'answer')
    assert q.get_text() == 'question'
    assert q.get_answer() == 'answer'
    assert q.ask_question() == 'question?'
    assert not (q.check_answer('ans'))
    assert q.check_answer('answer')

    q = FillInTheBlack('question', 'answer')
    assert q.get_text() == 'question'
    assert q.get_answer() == 'answer'
    assert q.ask_question() == 'question\nFill in the blank.'
    assert not (q.check_answer('ans'))
    assert q.check_answer('answer')

    q = TrueFalse('question', False)
    assert q.get_text() == 'question'
    assert q.get_answer() == 'False'
    assert q.ask_question() == 'question\nIs this statement true or false?'

    assert not (q.check_answer('random_ans'))
    assert q.check_answer('False')
    assert q.check_answer('false')

    print('All ShortAnswer tests passed!')
```



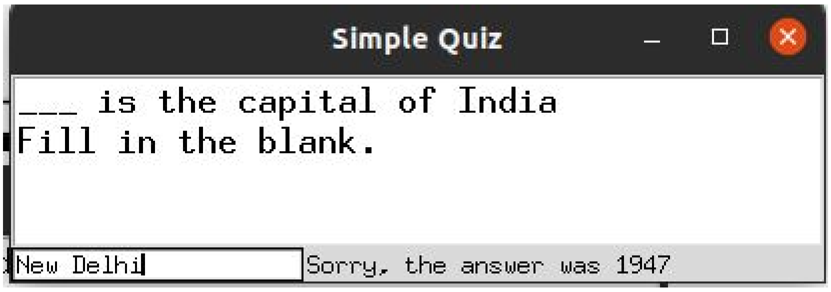
```
Sky is Blue
Is this statement true or false?
```

```
true    Good job! Next question ...
```



```
Independence Day of India?
```

```
1974    Good job! Next question ...
```



___ is the capital of India
Fill in the blank.

New Delhi

Sorry, the answer was 1947