PYTHON

LAB REPORT

ASSIGNMENT 1

1. Write a prime generator program using only primes and using python loops.

Code

from math import sqrt

def is\_prime(n):

if (n <= 1):

return False

if (n == 2):

return True

if (n % 2 == 0):

return False

i = 3

while i <= sqrt(n):

if n % i == 0:

return False

i = i + 2

return True

def prime\_generator():

n = 1

while True:

n += 1

if is\_prime(n):

yield n

generator = prime\_generator()

for i in range(5):

print(next(generator))

Output

Enter the number : 11

Prime : 2

Prime : 3

Prime : 5

Prime : 7

Prime : 11

2. Write a discount coupon code using dictionary in Python with different rate coupons for

each day of the week.

Code

Coupon = {'Monday' : 12,

'Tuesday' : 10,

'Wednesday' : 25,

'Thursday' : 10,

'Friday' : 20,

'Saturday' : 12,

'Sunday' : 5,

}

day = input("Enter the day of the week : ")

print("Discount for ",day," is : ",Coupon[day],"%")

Output

Enter the day of week : Sunday

Discount for Sunday is : 5%

3. Print first 10 odd and even numbers using iterators and compress.

Code

N = 10

iter\_odd = iter([(2\*i - 1) for i in range(1, N + 1)])

iter\_even = iter([(2\*i) for i in range(1, N + 1)])

print("Odd :", end=' ')

while True:

try:

odd\_item = next(iter\_odd)

print(odd\_item, end=' ')

except StopIteration:

break

print("\nEven :", end=' ')

while True:

try:

even\_item = next(iter\_even)

print(even\_item, end=' ')

except StopIteration:

break

print("")

Output

Odd : 1 3 5 7 9 11 13 15 17 19

Even : 2 4 6 8 10 12 14 16 18 20

4. Print the permutations of ABCDE using iterators.

Code

from itertools import permutations

str = "ABCDE"

p = permutations(str)

for j in list(p):

print(''.join(j),end=" ")

Output

ABCDE

ABCED

ABDCE

ABDEC

ABECD

ABEDC

...

EDBAC

EDBCA

EDCAB

EDCBA

5. Write a matrix multiplication function to compute.

Code

A = [[7, 8, 9], [1, 4, 7], [6, 4, 6]]

B = [[7, 7, 4, 7], [0, 5, 5, 4], [0, 1, 1, 5]]

def mat(n):

lst = []

for i in range(n):

temp = []

for j in range(n):

ele = int(input("Enter element : "))

temp.append(ele)

lst.append(temp)

return lst

n = int(input("Enter the number of rows and columns : "))

A = mat(n)

B = mat(n)

result = [[sum(a \* b for a, b in zip(A\_row, B\_col))

for B\_col in zip(\*B)]

for A\_row in A]

for r in result:

print(r)

Output

[49, 98, 77, 126]

[7, 34, 31, 58]

[42, 68, 50, 88]

6. Create list of servers, IP addresses and ports using variable positional and keyword

arguments.

Code

lst1 = ['192.168.1.1', '192.168.10.50' ,'192.168.20.1 ','10.0.0.138']

lst2 = ['25' , '80' , '443' ,' 3000']

servers = []

IP = []

ports = []

def assign(ip,\*args,port):

local = []

local.append(ip)

local.append(port)

for arg in args :

servers.append(arg)

return local

IP = assign(lst1,'Cold Fusion','Enhydra' , 'Jetty',port = lst2 )[0]

ports = assign(lst1,'Cold Fusion','Enhydra' , 'Jetty',port = lst2 )[1]

print(servers)

print(IP)

print(ports)

Output

[{'name': 'flask:localhost', 'ip': '127.0.0.1', 'port': '8080'}, {'name': 'nodejs:localhost', 'ip': '127.0.0.1', 'port': '3000'}, {'name': 'google.com', 'ip': None, 'port': None}]

7. Compute sorted five numbers using keyword-only arguments.

Code

def main():

usr\_input = input("Enter 5 numbers : ")

integers\_list = list(map(int, usr\_input.split()))

print(sorted(integers\_list, reverse=False))

if \_\_name\_\_=="\_\_main\_\_":

main()

Output

Enter 5 numbers : 5 8 7 6 2

[2, 5, 6, 7, 8]

8. Create a list of all the numbers up to N=50 which are multiples of five using anonymous

function.

Code

li = [x for x in range(51)]

final\_list = list(filter(lambda x : (x%5==0),li))

print(final\_list)

Output

[0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50]

9. Use map and zip to find the element-wise maximum amongst sequences of student list,

subject list and marks list.

Code

student\_names = ['Lisa', 'Rose', 'Jennie', 'Jisoo']

student\_marks = ['78', '54', '99', '98']

student\_subjects = ['OOPS', 'DLD', 'CO', 'DS & Algo']

def main():

tuples = list(zip(student\_names, student\_subjects, student\_marks))

maximum = sorted(tuples, key=lambda x: x[2], reverse=True)[0]

print(maximum)

if \_\_name\_\_=="\_\_main\_\_":

main()

Output

('Jennie', 'CO', '99')

10. Filter out the odd squares using map, filter, list.

Code

from math import sqrt

lst = [4,9,16,25,36,49,64,81,100]

final\_list\_1= list(filter(lambda x :sqrt(x)%2!=0,lst))

final\_list\_2 = list(map(lambda x :sqrt(x)%2,lst))

print(final\_list\_1)

print(final\_list\_2)

Output

[9,25,49,81]

[4,16,36,64,100]

11. Let's find all Pythagorean triples whose short sides are numbers smaller than 10. use

filter and comprehension.

Code

limit = 5

def get\_pythagorean\_triplets(m, n):

a = m\*\*2 - n\*\*2

b = 2 \* m \* n

c = m\*\*2 + n\*\*2

return sorted([a, b, c])

triplets = []

for n in range(1, 11):

for m in range(n + 1, 11):

triplets.append(get\_pythagorean\_triplets(m, n))

triplets\_shorter\_side\_less\_than\_10 = list(filter(lambda x: x[0] < 10, triplets))

print(triplets\_shorter\_side\_less\_than\_10)

Output

[[3, 4, 5], [6, 8, 10], [8, 15, 17], [5, 12, 13], [7, 24, 25], [9, 40, 41]]

12. Enumerate the sequence of all lowercase ASCII letters, starting from 1, using

enumerate.

Code

def main():

lowercase\_letters = [chr(x) for x in range(ord('a'), ord('z') + 1)]

enumerate\_lowercase = {}

for i, ch in enumerate(lowercase\_letters):

enumerate\_lowercase[i + 1] = ch

print(enumerate\_lowercase)

if \_\_name\_\_=="\_\_main\_\_":

main()

Output

{1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e', 6: 'f', 7: 'g', 8: 'h', 9: 'i', 10: 'j', 11: 'k', 12: 'l', 13: 'm', 14: 'n', 15: 'o', 16: 'p', 17: 'q', 18: 'r', 19: 's', 20: 't', 21: 'u', 22: 'v', 23: 'w', 24: 'x', 25: 'y', 26: 'z'}

13. Create a dictionary with comprehension with keys = the letters in the string of your

name, and values of the same letters, but with the case swapped.

Code

def get\_dictionary\_containing\_case\_swapped\_ascii\_values(name):

dictionary = {}

characters = list(name)

for ch in characters:

if ch in dictionary:

continue

if ch.isupper():

ch\_case\_swapped = ch.lower()

else:

ch\_case\_swapped = ch.upper()

dictionary[ch] = ord(ch\_case\_swapped)

return dictionary

def main():

str\_input = input("Enter your name : ").rstrip()

print(get\_dictionary\_containing\_case\_swapped\_ascii\_values(str\_input))

if \_\_name\_\_=="\_\_main\_\_":

main()

Output

Enter your name : Sandy

{'S': 115, 'a': 65, 'n': 78, 'd': 68, 'y': 89}

14. Write a python program to

1. read lines from a file, break into tokens and convert the tokens to unique numerical

values using python dictionary.

Code

F = 65537

import random

def generate\_hash(string, token\_dict):

if string in token\_dict:

return token\_dict[string]

characters = list(string)

char\_sum = 0

for ch in characters:

char\_sum += ord(ch) \* random.randint(0, 1e6)

return char\_sum % F

def main():

file = open("./14.txt", 'r')

lines = file.readlines()

file.close()

token\_dict = {}

for line in lines:

line = line.replace('\n', '')

words = line.split(' ')

for word in words:

hash = generate\_hash(word, token\_dict)

token\_dict[word] = hash

print(token\_dict)

if \_\_name\_\_=="\_\_main\_\_":

main()

2. Convert lines of different lengths into lines of same length (maximum length). Use

padding if and when required.

Code

LINE\_SIZE = 50

PAD\_CHARACTER = '-'

def main():

file = open("./14.txt", 'r')

lines = file.readlines()

file.close()

total\_string = ''.join(lines).replace('\n', ' ')

words = total\_string.split(' ')

file = open("./14\_2\_Edited.txt", 'w')

buffer = ""

for word in words:

temp\_buffer = buffer + word + " "

if len(temp\_buffer) < LINE\_SIZE:

buffer = temp\_buffer

else:

buffer += (LINE\_SIZE - len(buffer)) \* PAD\_CHARACTER

file.write(buffer + '\n')

buffer = word + " "

if len(buffer):

file.write(buffer + '\n')

file.close()

if \_\_name\_\_=="\_\_main\_\_":

main()

15. Write a python program to identify and extract numerical chunks from a text file and

convert them into words; e.g.; 1992 “nineteen hundred and ninety two”.

Code

def convert\_to\_words(num):

output\_string = ""

# Get number of digits

# in given number

l = len(num)

# Base cases

if (l == 0):

print("Empty String")

return

if (l > 4):

print("Length more than 4 is not supported")

return

# The first string is not used,

# it is to make array indexing simple

single\_digits = ["zero", "one", "two", "three",

"four", "five", "six", "seven",

"eight", "nine"]

# The first string is not used,

# it is to make array indexing simple

two\_digits = ["", "ten", "eleven", "twelve",

"thirteen", "fourteen", "fifteen",

"sixteen", "seventeen", "eighteen",

"nineteen"]

# The first two string are not used,

# they are to make array indexing simple

tens\_multiple = ["", "", "twenty", "thirty", "forty",

"fifty", "sixty", "seventy", "eighty",

"ninety"]

tens\_power = ["hundred", "thousand"]

# For single digit number

if (l == 1):

output\_string += (single\_digits[ord(num[0]) - '0'])

return output\_string

# Iterate while num is not '\0'

x = 0

while (x < len(num)):

# Code path for first 2 digits

if (l >= 3):

if (ord(num[x]) - 48 != 0):

output\_string += (single\_digits[ord(num[x]) - 48]) + " "

output\_string += (tens\_power[l - 3]) + " "

# here len can be 3 or 4

l -= 1

# Code path for last 2 digits

else:

# Need to explicitly handle

# 10-19. Sum of the two digits

# is used as index of "two\_digits"

# array of strings

if (ord(num[x]) - 48 == 1):

sum = (ord(num[x]) - 48 +

ord(num[x+1]) - 48)

output\_string += (two\_digits[sum])

return output\_string

# Need to explicitly handle 20

elif (ord(num[x]) - 48 == 2 and

ord(num[x + 1]) - 48 == 0):

output\_string += ("twenty")

return output\_string

# Rest of the two digit

# numbers i.e., 21 to 99

else:

i = ord(num[x]) - 48

if(i > 0):

output\_string += (tens\_multiple[i]) + " "

else:

output\_string += ("") + " "

x += 1

if(ord(num[x]) - 48 != 0):

output\_string += (single\_digits[ord(num[x]) - 48])

x += 1

return output\_string

def main():

file = open("./15.txt", 'r')

lines = file.readlines()

file.close()

file = open("./15\_Edited.txt", 'w')

for line in lines:

words = line.replace('\n', '').rstrip().split(' ')

buffer = ""

for word in words:

if word[0].isdigit():

cnvt\_word = convert\_to\_words(word)

buffer += cnvt\_word + " "

else:

buffer += word + " "

buffer = buffer.rstrip()

buffer = buffer[0].upper() + buffer[1:]

file.write(buffer + '\n')

file.close()

if \_\_name\_\_=="\_\_main\_\_":

main()

Output

My name is Tom and I am 20 years old.

42 is the answer to the ultimate question of life, the universe, and everything.

My name is Tom and I am twenty years old.

Forty two is the answer to the ultimate question of life, the universe, and everything.

ASSIGNMENT 2

1. Compare uppercase and lowercase of your name using set() and {} syntax.

Code

name = input("Enter name : ")

original = set(name)

upper = set(name.upper())

lower = set(name.lower())

print("Unique uppercase chars = " + str(original - lower))

print("Unique lowercase chars = " + str(original - upper))

Output

Enter name : SanDy

Unique uppercase chars = {'D', 'S'}

Unique lowercase chars = {'n', 'y', 'a'}

2. Write first seven Fibonacci numbers using generator next function/ yield in python.

Code

def fib():

a = 0

b = 1

i = 1

while i<=7 :

c = a + b

a = b

b = c

yield b

i = i + 1

fibonacci = fib()

while True:

try:

print(next(fibonacci),end = " ")

except StopIteration:

break

Output

1 2 3 5 8 13 21

3. Write a code which yields all terms of the geometric progression a, aq, aq2 , aq3 , .... 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.

Code

import time

def gp(base,cm=1):

series = []

x = time.time()

while base < 100000 :

series.append(base)

base \*= cm

y = time.time()

print("Time inside loop" + str(y-x) + "seconds")

return series

start = time.time()

a = int(input("Enter the value of a: "))

q = int(input("Enter the value of q: "))

if(q==1):

print("Infinite series")

else:

print(gp(a,q))

end = time.time()

print("Total time : " + str(end - start) + "seconds")

Output

1. Enter the value of a: 2

Enter the value of q: 3

Time inside loop2.8371810913085938e-05seconds

[2, 6, 18, 54, 162, 486, 1458, 4374, 13122, 39366]

Total time : 5.269235134124756seconds

4. Create a generator expression for the first 10 cubes.

Code

cubes = (num \*\* 3 for num in range(1, 11))

for cube in cubes:

print(cube, end=" ")

Output

1 8 27 64 125 216 343 512 729 1000

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

Code

class Square:

def \_\_init\_\_(self, side):

self.\_\_side = side

def calcArea(self):

return self.\_\_side \*\* 2

s = int(input("Enter side : "))

sq = Square(s)

print("Area : {}".format(sq.calcArea()))

Output

Enter side : 7

Area : 49

6. Create book, ebook, journal classes to use inheritance with title, publisher, page, year of publishing details.

Code

class Book:

def \_\_init\_\_(self, title, cat, year):

self.title = title

self.category = cat

self.year = year

def changeTitle(self):

self.title = input("Enter new title : ")

class EBook(Book):

def \_\_init\_\_(self, title, year, pages):

super().\_\_init\_\_(title, "EBook", year)

self.pages = pages

def changePageCount(self):

self.pages = input("Enter new page count : ")

class Journal(Book):

def \_\_init\_\_(self, title, year, publisher):

super().\_\_init\_\_(title, "Journal", year)

self.publisher = publisher

def changePageCount(self):

self.publisher = input("Enter new page count : ")

category = int(input("Enter the category of book (1 - EBook, 2 - Journal) : "))

if category == 1:

t = input("Enter title of the e-book : ")

y = input("Enter year of publishing of the e-book : ")

p = input("Enter publisher of the e-book : ")

obj = EBook(t, y, p)

elif category == 2:

t = input("Enter title of the journal : ")

y = input("Enter year of publishing of the journal : ")

cnt = input("Enter page count of the journal : ")

obj = EBook(t, y, cnt)

else:

print("Wrong Choice !!")

Output

1. Enter the category of book (1 - EBook, 2 - Journal) : 1

Enter title of the e-book : Django for Beginners

Enter year of publishing of the e-book : 2017

Enter publisher of the e-book : Wiley

1. Enter the category of book (1 - EBook, 2 - Journal) : 3

Wrong Choice !!

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

Code

name = input("Enter your name : ")

# storing the characters of the given name in title case

base = set(name)

# converting name to uppercase

uppercase = set(name.upper())

# converting name to lowercase

lowercase = set(name.lower())

print("Unique uppercase characters : " + str(base - lowercase))

print("Unique lowercase characters : " + str(base - uppercase))

Output

Enter your name : Aritra Samanta

Unique uppercase characters : {'S', 'A'}

Unique lowercase characters : {'t', 'i', 'r', 'n', 'm', 'a'}

8. Search for palindrome and unique words in a text using class method and string method.

Code

class Shape:

def \_str\_(self):

return "Base class, inheritance is needed for advanced use."

class SmoothShape(Shape):

def \_str\_(self):

return super().\_str\_()

class NonSmoothShape(Shape):

def \_str\_(self):

return super().\_str\_()

class TwoDimensionalShape(Shape):

def \_str\_(self):

return super().\_str\_()

class ThreeDimensionalShape(Shape):

def \_str\_(self):

return super().\_str\_()

class RegularPolygon(TwoDimensionalShape, NonSmoothShape):

def \_init\_(self, length, sides = 3):

self.sides = sides

self.length = length

def \_getArea(self):

perimeter = self.sides \* self.length

apothem = round(self.length / (2 \* math.tan(math.pi/self.sides)), 3)

area = (perimeter \* apothem) / 2

return area

def getArea(self):

print("Area of regular polygon: {:.3f}".format(self.\_getArea()))

class Square(RegularPolygon):

def \_init\_(self, length):

super().\_init\_(length, 4)

def getArea(self):

print("Area of square: {:.3f}".format(super().\_getArea()))

class Rectangle(TwoDimensionalShape, NonSmoothShape):

def \_init\_(self, length, breadth):

self.length = length

self.breadth = breadth

def getArea(self):

print("Area of rectangle: {:.3f}".format(self.length \* self.breadth))

class Circle(TwoDimensionalShape, SmoothShape):

def \_init\_(self, radius):

self.radius = radius

def getArea(self):

print("Area of circle: {:.3f}".format(math.pi \* (self.radius \*\* 2)))

class Cuboid(ThreeDimensionalShape, NonSmoothShape):

def \_init\_(self, length, breadth, height):

self.length = length

self.breadth = breadth

self.height = height

def getArea(self):

area = 2 \* (self.length \* self.breadth + self.length \* self.height + self.height \* self.breadth)

print("Area of cuboid: {:.3f}".format(area))

class Cube(ThreeDimensionalShape, NonSmoothShape):

def \_init\_(self, length):

self.length = length

def getArea(self):

area = 6 \* (self.length \*\* 2)

print("Area of cube: {:.3f}".format(area))

class Cone(ThreeDimensionalShape, NonSmoothShape):

def \_init\_(self, height, base\_radius):

self.height = height

self.base\_radius = base\_radius

def getArea(self):

area = (math.pi \* self.base\_radius \* 2) + math.pi \* self.base\_radius \* math.sqrt((self.base\_radius \* 2) + (self.height \*\* 2))

print("Area of cone: {:.3f}".format(area))

9. Check and set a person's age in person class using property decorator.

Code

class Person:

def \_init\_(self):

self.\_name = ''

self.\_age = -1

@property

def name(self):

return self.\_name

@property

def age(self):

return self.\_age

@name.setter

def name(self, value):

self.\_name = value

@name.deleter

def delete(self):

print("Deleting name property")

del self.\_name

@age.setter

def name(self, value):

self.\_age = value

@age.deleter

def delete(self):

print("Deleting name property")

del self.\_age

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

Code

class String:

def \_\_init\_\_(self, string):

self.string = string

def \_\_len\_\_(self):

length = 0

for word in self.string.split()[-3:]:

length += len(word)

return length

s = String(input("Enter a sentence : "))

print("Output from modified len : {}".format(len(s)))

Output

Enter a sentence : Hello I am 3 yrs old

Output from modified len : 7

11. Write an 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.

Code

from collections import Counter

def findLen(string):

counter = 0

for i in string:

counter += 1

return counter

def len(string) -> list:

length = []

total = 0

count = Counter()

for word in string.split():

if count.get(word) is None:

count[word] = 1

total += 1

else:

count[word] += 1

for word, cnt in count.most\_common(total):

if cnt == 1:

break

length.append((word, findLen(word)))

return length

print(len(input("Enter a sentence : ")))

Output

Enter a sentence : Eat Sleep Code Repeat Eat Think Code Repeat Code More Code Stop

[('Code', 4), ('Eat', 3), ('Repeat', 6)]

12.Write a function that flattens a nested dictionary structure like one obtained from Twitter and Facebook APIs or from some JSON file.  
   
nested = {  
 'fullname': 'Alessandra',  
 'age': 41,  
 'phone-numbers': ['+447421234567', '+447423456789'], 'residence': {  
   
'address': {  
 'first-line': 'Alexandra Rd',  
 'second-line': ''

},

'zip': 'N8 0PP', 'city': 'London', 'country': 'UK', }

}

Code

import json

import pprint

content = open('api.json').read()

content = json.loads(content)

flattened ={}

def check\_type(x):

if type(x) == int :

return True

if type(x) == float:

return True

if type(x) == str:

return True

def rec( tmp, key\_name):

if check\_type(tmp):

flattened[key\_name] = tmp

return

if type(tmp) == list or type(tmp) == tuple:

for i , item in enumerate(tmp):

rec( item , key\_name+'\_'+str(i))

else:

for key in tmp.keys():

rec( tmp[key] , key\_name + '\_' + str(key) )

pprint.pprint('Content : ' , content)

rec( content , '')

# print(content)

pprint.pprint('Flattened : ' , flattened)

Output

Content :{'age': 41,

'fullname': 'Alessandra',

'phone-numbers': ['+447421234567', '+447423456789'],

'residence': {'address': {'first-line': 'Alexandra Rd', 'second-line': ''},

'city': 'London',

'country': 'UK',

'zip': 'N8 0PP'}}

Flattened : {'\_age': 41,

'\_fullname': 'Alessandra',

'\_phone-numbers\_0': '+447421234567',

'\_phone-numbers\_1': '+447423456789',

'\_residence\_address\_first-line': 'Alexandra Rd',

'\_residence\_address\_second-line': '',

'\_residence\_city': 'London',

'\_residence\_country': 'UK',

'\_residence\_zip': 'N8 0PP'}

13. 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.

Code

# Q13.py

from nose.tools import assert\_equal

from nose\_parameterized import parameterized

import math

@parameterized([

(2, 2, 4),

(2, 3, 8),

(1, 9, 1),

(0, 9, 0),

])

def test\_pow(base, exponent, expected):

assert\_equal(math.pow(base, exponent), expected)

Output

$ py.test -v test\_math.py

========================= test session starts ==========================

platform darwin -- Python 3.7.3 -- py-1.4.30 -- pytest-2.7.1

collected 4 items

test\_math.py::test\_pow::[0] PASSED

test\_math.py::test\_pow::[1] PASSED

test\_math.py::test\_pow::[2] PASSED

test\_math.py::test\_pow::[3] PASSED

========================= 4 passed in 0.07 seconds =====================

14.Use profile/cprofile to check pythagorean triples code in python. Think about time complexity of the code.

Code

import cProfile

def gen\_triplets(limit=None):

limit = limit if limit is not None else 20

c = 0

m =2

while c < limit:

for n in range( 1 , m ):

a = m\*m - n\*n

b = 2 \* m \* n

c = m\*m + n\*n

if c > limit:

break

print( a , b , c )

m+=1

cProfile.run('gen\_triplets()')

Output

3 4 5

8 6 10

5 12 13

15 8 17

12 16 20

9 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 q14.py:3(gen\_triplets)

1 0.000 0.000 0.000 0.000 {built-in method builtins.exec}

5 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}

15. 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.

Code

class Student:

def \_\_init\_\_(self, name, score1, score2, score3):

self.name = name

self.score1 = score1

self.score2 = score2

self.score3 = score3

def getName(self):

return self.name

def calcTotal(self):

return self.score1 + self.score2 + self.score3

class MarkSheet:

def \_\_init\_\_(self, students):

self.results = {}

for each in students:

self.results[each] = each.calcTotal()

def showRank(self):

for ranker in sorted(self.results.items(), key=lambda kv: (kv[1], kv[0]), reverse=True):

print(ranker[0].getName() + " : " + str(ranker[1]))

batch = []

while True:

identity = input("Enter student name : ")

marks = input("Enter marks in 3 subjects : ").split()

batch.append(Student(identity, int(marks[0]), int(marks[1]), int(marks[2])))

check = int(input("Do you want to add more students ? (1 -> Yes, 0 -> No) : "))

if check == 0:

break

markSheet = MarkSheet(batch)

markSheet.showRank()

Output

Enter student name : Sandipan Saha

Enter marks in 3 subjects : 98 97 100

Do you want to add more students ? (1 -> Yes, 0 -> No) : 1

Enter student name : Mritunjoy Dey

Enter marks in 3 subjects : 96 99 97

Do you want to add more students ? (1 -> Yes, 0 -> No) : 1

Enter student name : Shoif Md Mia

Enter marks in 3 subjects : 35 67 41

Do you want to add more students ? (1 -> Yes, 0 -> No) : 1

Enter student name : Sk Sabbir Hossain

Enter marks in 3 subjects : 93 99 95

Do you want to add more students ? (1 -> Yes, 0 -> No) : 0

Sandipan Saha : 295

Mritunjoy Dey : 292

Shoif Md Mia : 287

Sk Sabbir Hossain : 143

16.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)

Code

s1 = "always"

s2 = "ways"

table = [[1e9 for \_ in range( len(s2) +1 )] for \_ in range(len(s1) +1 )]

par = [[None for \_ in range( len(s2) +1 )] for \_ in range(len(s1) +1 )]

for i in range( 0, len(s1)+1):

for j in range( 0 , len(s2) + 1) :

if i == 0 :

table[i][j] = j

elif j == 0 :

table[i][j] = i

elif s1[i-1] == s2[j-1]:

table[i][j] = table[i-1][j-1]

par[i][j] = [ i -1, j-1 ]

else:

table[i][j] = min( [table[i-1][j-1] , table[i-1][j] , table[i][j-1]]) +1

if table[i][j] == table[i-1][j-1]:

par[i][j] = [ i -1 , j-1 ]

if table[i][j] == table[i][j-1]:

par[i][j] = [i , j-1]

if table[i][j] == table[i-1][j]:

par[i][j] = [i - 1, j]

i , j = len(s1), len(s2)

print('Minimum changes required :' , table[len(s1)][len(s2)])

# down - delete

# left - insert

# diagonal - substitute

def decode( prev , next) :

if prev[0] == next[0] and next[1] + 1 == prev[1]:

# left

return 'I'

elif prev[1] == next[1] and next[0] + 1 == prev[0]:

# down

return 'D'

else:

# substitute

return 'S'

actions =""

while par[i][j] is not None :

prev = [i , j ]

i , j = par[i][j]

actions += decode( (i , j ) , prev)

if len(actions) != len(s1):

if i == 0:

actions += ''.join('I' for \_ in range(len(s2) - len(s1)))

elif j == 0:

actions += ''.join('D' for \_ in range( len(s1) - len(s2)))

actions = actions[::-1]

print('s1 = ' , s1)

print('s2 = ' , s2)

print('Action list to get from s1 to s2 D(for delete) S(for substitute) and I (for insert):\n' , actions)

Output

Minimum changes required : 2

s1 = always

s2 = ways

Action list to get from s1 to s2 D(for delete) S(for substitute) and I (for insert):

DDSSSS

17.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).

Code

import re

def decontracted(phrase):

# specific

phrase = re.sub(r"won\'t", "will not", phrase)

phrase = re.sub(r"can\'t", "can not", phrase)

# general

phrase = re.sub(r"n\'t", " not", phrase)

phrase = re.sub(r"\'re", " are", phrase)

phrase = re.sub(r"\'s", " is", phrase)

phrase = re.sub(r"\'d", " would", phrase)

phrase = re.sub(r"\'ll", " will", phrase)

phrase = re.sub(r"\'t", " not", phrase)

phrase = re.sub(r"\'ve", " have", phrase)

phrase = re.sub(r"\'m", " am", phrase)

return phrase

def multi\_to\_single(phrase):

phrase = re.sub(r"[\s]+" , " " , phrase)

return phrase

def remove\_special(phrase):

phrase = re.sub(r"[^A-Za-z0-9\s]+" , "" , phrase)

return phrase

def remove\_single(phrase):

phrase = re.sub(r"(^[a-zA-z]\s)|(\s[a-zA-z]\s)|(\s[a-zA-z]$)", " ", phrase)

return phrase

def convert\_lower(phrase):

phrase = re.sub( r"[A-Z]" , lambda m : m.group().lower() , phrase )

return phrase

phrase = "hello long spaces did it work?"

print(phrase , '\n :after removing mutiple spaces')

print( multi\_to\_single(phrase))

phrase ="A perfectly normal sentence : oops \*\* omg ^(& let's clean it up "

print(phrase, '\n :after removing special characters')

print(multi\_to\_single(remove\_special(phrase)))

phrase = "This sentence is going to be weird without a 'a' b s cd d"

print(phrase, '\n :after removing single characters')

print(remove\_single(phrase))

phrase = "A phrase containing Upper and Lower CAses"

print(phrase, '\n :after converting into lower cases')

print(convert\_lower(phrase))

phrase="I've always loved using decontracted words and i'll continue"

print(phrase, '\n :after decontracting')

print(decontracted(phrase))

Output

hello long spaces did it work?

:after removing mutiple spaces

hello long spaces did it work?

A perfectly normal sentence : oops \*\* omg ^(& let's clean it up

:after removing special characters

A perfectly normal sentence oops omg lets clean it up

This sentence is going to be weird without a 'a' b s cd d

:after removing single characters

This sentence is going to be weird without 'a' s cd

A phrase containing Upper and Lower CAses

:after converting into lower cases

a phrase containing upper and lower cases

I've always loved using decontracted words and i'll continue

:after decontracting

I have always loved using decontracted words and i will continue

18.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.

Code

import numpy as np

a = np.random.randint( 0 ,20 , 15 )

print('The vector : ' , \*a )

print('Most frequent element : ' , np.argmax(np.bincount(a)))

binc = np.bincount(a)

## masking to prevent counting non-present items

binc[binc == 0] = 1e9

# print(binc)

print('Least frequent element : ', np.argmin(binc))

Output

The vector : 17 11 11 13 7 17 16 17 3 12 19 6 7 2 4

Most frequent element : 17

Least frequent element : 2

19.Check http://yann.lecun.com/exdb/mnist/ web page. Execute the training-testing model of classifications and compare accuracy and other ROC measures for the classification solutions for any two algorithms using Python among - K-NN with non-linear deformation (IDM), K-NN, shape context matching, Y. LeCun, L. Bottou and Y. Bengio: Reading Checks with graph transformer networks, 3-layer NN, 500+300 HU, softmax, cross entropy, weight decay, 6-layer NN 784-2500- 2000-1500-1000-500-10 (on GPU) [elastic distortions], Convolutional net Boosted LeNet-4, [distortions], committee of 35 conv. net, 1-20-P-40-P-150-10 [elastic distortions].

Code

from sklearn.metrics import confusion\_matrix , accuracy\_score

from sklearn.neural\_network import MLPClassifier

import matplotlib.pyplot as plt

import numpy as np

import gzip

import random

from sklearn.metrics import roc\_auc\_score

import time

from sklearn.neighbors import KNeighborsClassifier

f = gzip.open('train-images-idx3-ubyte.gz', 'r')

image\_size = 28

num\_images = 60000

f.read(16)

buf = f.read(image\_size \* image\_size \* num\_images)

data = np.frombuffer(buf, dtype=np.uint8).astype(np.float32)

train\_data = data.reshape(num\_images, image\_size \* image\_size)

# image = np.asarray(data[random.choice(list(range(num\_images)))])

# plt.imshow(image)

# plt.show()

f = gzip.open('train-labels-idx1-ubyte.gz', 'r')

f.read(8)

buf = f.read( num\_images )

data = np.frombuffer(buf, dtype=np.uint8).astype(np.float32)

train\_labels = data.reshape( num\_images )

# print(train\_labels[random.choice(list(range(num\_images)))])

f = gzip.open('t10k-labels-idx1-ubyte.gz', 'r')

test\_num\_images = 10000

f.read(8)

buf = f.read( test\_num\_images )

data = np.frombuffer(buf, dtype=np.uint8).astype(np.float32)

test\_labels = data.reshape( test\_num\_images )

# print(train\_labels[random.choice(list(range(test\_num\_images)))])

f = gzip.open('t10k-images-idx3-ubyte.gz', 'r')

f.read(16)

buf = f.read(image\_size \* image\_size \* test\_num\_images)

data = np.frombuffer(buf, dtype=np.uint8).astype(np.float32)

test\_data = data.reshape(test\_num\_images, image\_size \* image\_size)

def train\_knn():

st = time.time()

classifier = KNeighborsClassifier(n\_neighbors=7)

classifier.fit(train\_data, train\_labels)

pred = classifier.predict(test\_data)

en = time.time() - st

return pred , en

def train\_3\_layer\_nn():

st = time.time()

classifier = MLPClassifier( hidden\_layer\_sizes=(500,300) )

classifier.fit(train\_data , train\_labels)

pred = classifier.predict(test\_data)

en = time.time() - st

return pred ,en

method ="KNN"

pred, en = train\_knn()

# method = "3 Layer Neural Network"

# pred ,en = train\_3\_layer\_nn()

cm = confusion\_matrix(test\_labels, pred)

print(cm)

# print("7 neighbours")

print("Algorithm :" , method)

print( 'accuracy : ' , accuracy\_score( test\_labels , pred) )

Output

[[ 966 1 3 1 0 1 5 2 1 0]

[ 0 1130 0 2 0 1 1 0 1 0]

[ 0 1 1009 1 2 0 0 11 8 0]

[ 0 1 8 984 0 4 0 2 9 2]

[ 0 1 7 1 923 0 4 3 2 41]

[ 1 1 0 17 1 855 2 3 4 8]

[ 3 2 3 0 5 2 937 0 6 0]

[ 2 5 7 3 3 0 0 995 2 11]

[ 2 1 3 6 0 3 0 2 952 5]

[ 3 3 1 5 1 3 0 2 5 986]]

Algorithm : 3 Layer Neural Network

accuracy : 0.9737

[[ 974 1 1 0 0 1 2 1 0 0]

[ 0 1133 2 0 0 0 0 0 0 0]

[ 10 9 996 2 0 0 0 13 2 0]

[ 0 2 4 976 1 13 1 7 3 3]

[ 1 6 0 0 950 0 4 2 0 19]

[ 6 1 0 11 2 859 5 1 3 4]

[ 5 3 0 0 3 3 944 0 0 0]

[ 0 21 5 0 1 0 0 991 0 10]

[ 8 2 4 16 8 11 3 4 914 4]

[ 4 5 2 8 9 2 1 8 2 968]]

3 neighbours

Algorithm : KNN

accuracy : 0.9705

20.Create a quiz GUI using inheritance and polymorphism in Python to ask a sequence of questions of the user. You may follow controller.py, quiz.py or shortAnswer.py files or may write your own codes. The interface is shown here:https://cs.calvin.edu/courses/cs/108/vnorman/13oop/lab.html

Create a new package for this lab called Exercise20 and copy the starting code files into this package: controller.py, quiz.py, shortAnswer.py (files attached).

Familiarize yourself with the quiz mechanism by doing the following: Run the controller a couple times;  
Run the unit tests for the ShortAnswer problem class;  
Add a new short-answer problem (of your choice) to the quiz.  
   
Inheritance  
   
Right now this quiz mechanism can only ask short answer problems, but this is a bit too limited for the purposes. You will add fill-in-the-blank problems, true-false problems, and maybe even multiple choice problems. As we start planning, we realize that we will be duplicating code if we write each kind of problem from scratch. Instead, let's start with a Problem class that will be the parent class of all the different kinds of problems. This class will collect all of the attributes and methods that are shared between all problems.

So what is shared between all problems? Each problem has some text, but asks the question in a different way (e.g. short-answer just added a question mark, but a fill-in-the-blank problem should add both the question mark and an indication to "Fill in the blank."). Further, all problems have answers, but a true-false problem has a boolean answer instead of a string. Considering these properties, we proceed as follows:

Exercise 20.2

Refactor your code to include a Problem class and a ShortAnswer class that inherits from Problem. The new Problem class:

Includes a constructor that receives a string and stores it in an instance variable called self.text

Includes an accessor for the question called get\_text()

* Put this in a separate file. The updated ShortAnswer class:  
   Has an updated class declaration that indicates it is inheriting from Problem

Has an updated constructor:  
   
o Calls the Problem constructor to initialize the text instance variable (instead of doing the assignment statement itself): Problem.\_\_init\_\_(self, q)  
   
o Does not remove or change the initialization of the answer instance variable

* Has an updated ask\_question method that replaces the access of self.text with a call to the appropriate accessor in the Problem class and then appends the question  
     
  mark: self.get\_text() + '?'
* Removes the get\_text method (since the Problem class is taking care of this for us).  
     
  If all has gone well, your controller should run just as it did before, and all of the ShortAnswer  
     
  tests should still pass.  
   Python does not require that the question class definitions be placed in separate files, but it is common practice to separate more complex classes into separate files. These question classes are probably simple enough that they can be kept together, but if you choose this option, you should rename the file to “problem” to more accurately indicate what it contains.  
   Polymorphism  
   We are now ready to add more types of problems, such as true-false problems and fill-in-the-blank problems. Examples of how your application should present these questions are shown below:

The new problem classes are very similar to a ShortAnswer problem, but they each have differences:

The FillInTheBlank class is very similar to the ShortAnswer class (so similar we should maybe

not have a whole separate class, but this is for learning purposes, right?) with the following modification:

o The ask\_question method must append the string '\nFill in the blank.' to the end of the problem text instead of just a question mark.

The TrueFalse class is also similar, but with a few more details to be worked out:

o The answer is expected to be a boolean value, so the constructor should raise an exception if the calling program provides an answer that is not an instance of bool. You

can check that answer is boolean using: isinstance(answer, bool).

o The ask\_question method must append the string '\nIs this statement true or false?' to

the end of the problem text.

o The get\_answer method should return a string (to match the other

problems): str(self.answer)

o The check\_answer method will receive a string, so we must compare a string version of

the correct answer to the received answer.

Because all problems are using the same method names, the quiz will be able to create a list of Problems, and ask each problem to ask its question, check its answer, and tell us the correct answer.

Exercise 20.3

Do the following (if you haven't already)   
 Add the FillInTheBlank and TrueFalse classes as described above.   
 Add some unit tests to the test cases at the end of the file(s).  
   
 Add at least one sample problem for each of the two new classes to the quiz.  
   
Your quiz application should now operate as it did before, with randomly ordered questions of all three subtypes.

Problem class Implemented

'''

Models a problem base class to be inherited

'''

class Problem:

def \_\_init\_\_(self ,text) -> None:

self.text = text

def get\_text(self):

'''

returns the problem text

'''

return self.text

FillinTheBlanks class Implemented

from shortAnswer import ShortAnswer

class FillInTheBlanks( ShortAnswer):

def \_\_init\_\_(self ,q ,a ) -> None:

super( FillInTheBlanks , self ).\_\_init\_\_(q ,a)

def ask\_question( self ):

return self.get\_text() + '\nFill in the blanks'

if \_\_name\_\_ == "\_\_main\_\_":

q = FillInTheBlanks('Dibble means to \_\_\_\_ like a duck', 'quack')

assert q.get\_text() == 'Dibble means to \_\_\_\_ like a duck'

assert q.get\_answer() == 'quack'

assert q.ask\_question() == 'Dibble means to \_\_\_\_ like a duck\nFill in the blanks'

assert not (q.check\_answer('ans'))

assert q.check\_answer('quack')

print('All FillintheBlanks tests passed!')

TrueFalse class Implemented

from shortAnswer import ShortAnswer

class TrueFalse(ShortAnswer):

def \_\_init\_\_(self, q, a) -> None:

super(TrueFalse, self).\_\_init\_\_(q, a)

def ask\_question(self):

return self.get\_text() + '\nIs This statement True or False?'

def get\_answer(self):

return str(self.answer)

def check\_answer(self ,ans):

return str(ans) == str(self.answer)

if \_\_name\_\_ == "\_\_main\_\_":

q = TrueFalse('Fall 2014 was short', True)

assert q.get\_text() == 'Fall 2014 was short'

assert q.get\_answer() == 'True'

assert q.ask\_question() == 'Fall 2014 was short\nIs This statement True or False?'

assert not (q.check\_answer('ans'))

assert q.check\_answer(True)

print('All TrueFalse tests passed!')

Updated short Answer

from problem import Problem

class ShortAnswer(Problem):

''' Model a short-answer question '''

def \_\_init\_\_(self, q, a):

super(ShortAnswer ,self ).\_\_init\_\_(q)

''' Construct a short-answer question with question and answer texts '''

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

if \_\_name\_\_ == "\_\_main\_\_":

q = ShortAnswer('question', 'answer')

assert q.get\_text() == 'question'

assert q.get\_answer() == 'answer'

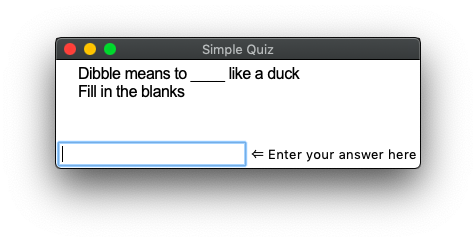
assert q.ask\_question() == 'question?'

assert not (q.check\_answer('ans'))

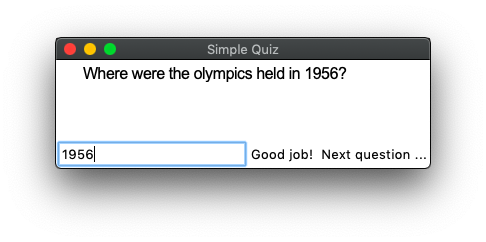
assert q.check\_answer('answer')

print('All ShortAnswer tests passed!')%

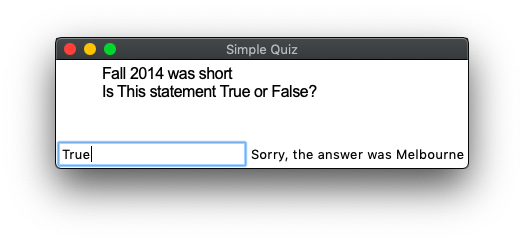
Output



Fill in the blank type quiz question



Short Answer type quiz question



True False type quiz question