**Python Tasks - Solutions**

# Task # 1

**Question**

Write a program which will find all such numbers which are divisible by 7 but are not a multiple of 5, between 2000 and 3200 (both included). The numbers obtained should be printed in a comma-separated sequence on a single line.

**Solution**

print("\*This program will print only the numbers between 2000-3500 which are divisible by 7 but not multiple of 5\*")

#creating list to save the numbers in the format of list later

numberlist = []

#loop to iterate 1200 times but within the range fo 1200 to 3200

for i in range (2000, 3200):

#check statement for divisible by 7 & not multiple of 5

if i%7==0 and i%5 !=0:

#adding/appending each approved value to list

numberlist.append(i)

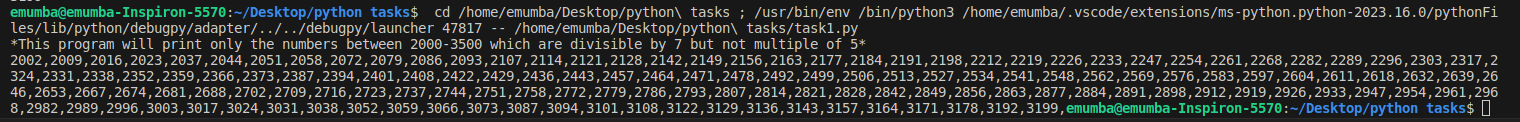
#printing the values

for numberlist in numberlist:

#printing the list with 'end' function to specify appending ',' at the end of each printed item

print(numberlist, end=",")

[**Link for the image**](https://drive.google.com/file/d/12S-T4GywBfhgnGrzXqW9ltOYqd2ZEK7b/view?usp=sharing)



# Task # 2

**Question**

The numbers after the direction are steps. Please write a program to compute the distance from the current position after a sequence of movement and original point. If the distance is a float, then just print the nearest integer. Use **argparse** library to take inputs for UP, DOWN, LEFT and RIGHT. Use of functions is encouraged.

Example: If the following tuples are given as input to the program:

UP 5

DOWN 3

LEFT 3

RIGHT 2

Then, the output of the program should be: 2

**Solution**

import argparse

import math

#func for user input

#creates argumentparser obj with description

def user\_input():

parser = argparse.ArgumentParser(description="Enter integer or floating-point values for the directions mentioned:")

#defining the list of arguments as directions value

directions\_value = ['int1','int2','int3','int4']

#loop for adding the arg names to argumentparser &

#rounding the value to int value if float is entered

for directions\_value in directions\_value:

parser.add\_argument(directions\_value, type=convert\_to\_nearest\_int)

#parsing for command line arguments

args = parser.parse\_args()

#extracting the values of int1,int2,int3,int4

int1 = args.int1

int2 = args.int2

int3 = args.int3

int4 = args.int4

#calling the func to calcualte the distances using the parsed argument

calculation(int1, int2, int3, int4)

# Custom type conversion function to convert to nearest integer if float is entered

def convert\_to\_nearest\_int(value):

try:

#rounding the called value in float to make it int

return round(float(value))

#return any anomly

except ValueError:

raise argparse.ArgumentTypeError(f"Invalid value: {value}. Please enter a valid number.")

#func to change any negative value to positive

def absolute\_value(value):

if isinstance(value,(int,float)):

return abs(value)

#calculation func using values

def calculation(int1, int2, int3, int4):

#

distance = int1 + int2 + int3 + int4

displacement1 = absolute\_value((int1 - int2))

displacement2 = absolute\_value((int3 - int4))

#value generated for the distance from final point to original position

disp = absolute\_value((math.sqrt((displacement1 \*\* 2) + (displacement2 \*\* 2))))

output(int1, int2, int3, int4, distance,disp)

def output(int1, int2, int3, int4, distance,disp):

print(f"UP {int1}")

print(f"DOWN {int2}")

print(f"LEFT {int3}")

print(f"RIGHT {int4}")

print(f"T. Traveled distance {convert\_to\_nearest\_int(distance)}")

print(f"distance from original position {convert\_to\_nearest\_int(disp)}")

if \_\_name\_\_ == '\_\_main\_\_':

user\_input()

# 

# 

# 

# 

# 

# 

# Task # 3

**Question**

Suppose you are a hardware enthusiast and love checking system’s details. To make your task easy you have to write a program that is used to check hardware details of a system and generates a file for you; named “Summary.txt '' at a location “/home/Username/Details”. If the directory “Details'' does not exist on your system you have to create it. Details you are interested in are given below along with example values. Remember you are not allowed to code in iPython. You can only use the Python3 interpreter. Username will be the name of the user on your system for example “/home/engrhamza/Details''

Byte Order: Little Endian

Core(s) per socket: 4

Socket(s): 1

Model name: Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz

CPU MHz: 1638.462

CPU max MHz: 4000.0000

CPU min MHz: 400.0000

Virtualization Support: VT-x

L1 32K

L2 cache: 256K

L3 cache: 8192K

RAM Memory: 15794MB

To make your problem easy you are allowed to add more details then the required one.

**Solution**

#importing module subprocess for allowing bash commands

#importing os module to interact with OS for various tasks

import subprocess

import os

#get username of this linux system using os module

get\_username=os.getlogin()

#defining the directory & its path(for txt file we need) inside a variable to be used later in file handling

store\_directory\_path = f"/home/{get\_username}/Details"

#defining the path with respect to the directory & file

file\_path = os.path.join(store\_directory\_path, "Summary.txt")

#creating a directory named "Details" if it doesnot exists

if not os.path.exists(store\_directory\_path):

os.makedirs(store\_directory\_path)

#bash commands for system information using lscpu and storing

# them into commands lists

command = ["lscpu | grep 'Byte Order'",

"lscpu | grep 'Core(s) per socket'",

"lscpu | grep 'Socket(s)'",

"lscpu | grep 'Model name'",

"lscpu | grep 'MHz'",

"lscpu | grep 'Virtualization'",

"lscpu | grep 'L1d cache' && lscpu | grep 'L1i cache'",

"lscpu | grep 'L2 cache'",

"lscpu | grep 'L3 cache'",

"free -h | awk '/Mem/ {print \"RAM Memory: \" $2}'"

]

#Opening and writing the generated results into Summary.txt' file

with open(file\_path, "w") as file\_path:

for command in command:

#executing each of the commands and storing inside 'result' var

result = subprocess.check\_output(command, shell=True, text=True)

#printing the reuslts on console

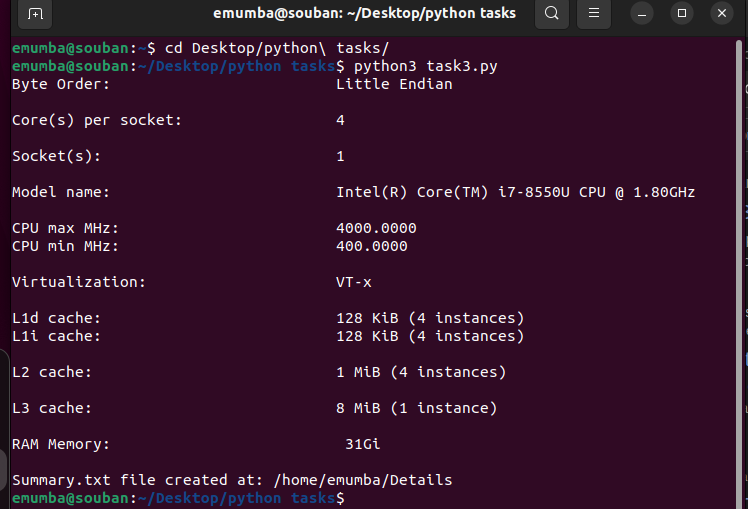
print(result)

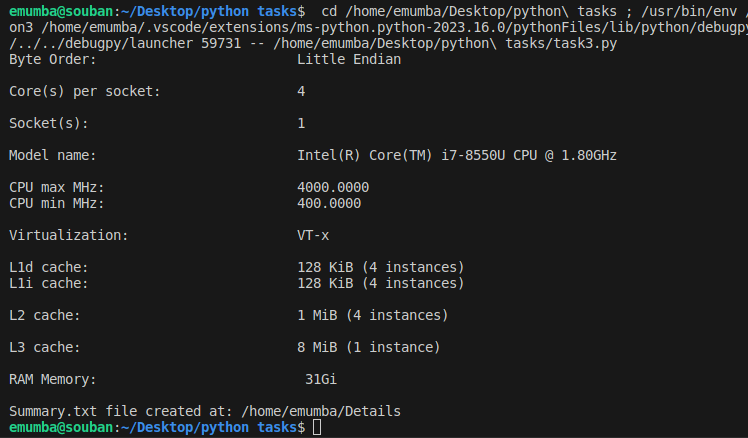
#writing the 'result' into the file path /home/{username}/Details/Summary.txt

file\_path.write(result)

#printing the completion message

print(f"Summary.txt file created at: {store\_directory\_path}")





Task # 4

**Question**

You have to implement Monte Carlo’s simulation for finding the value of “pi”? You can look up Monte Carlo’s simulation on Google. You have to take the number of iterations from the user. User needs to pass the “-i” flag for entering the value of iterations. If he or she enters the “-h” flag then print the help of your tool explaining what it does and what are the possible inputs. If the “-j” flag is passed then your program has to read the value of “iterations” from a JSON file placed in the same directory your script is placed. Make sure you use the argparse library for flags and JSON format for the JSON file.

Main.py

Project.css

Iterations.json

**Solution**

#importing 'random' for generating random numbers, 'argparse' to allow

#using of argument parsing through terminal, 'json' to let the code read json files

import random

import argparse

import json

#main code for generating random numbers (x,y) 7 calculating estimations

def estimate\_pi(iterations):

#variables for points inside circle and square

points\_inside\_circle = 0

points\_inside\_square = 0

#loop for

for \_ in range(iterations):

#generating random values between -1 to 1 in x and y directions

rand\_x = random.uniform(-1, 1)

rand\_y = random.uniform(-1, 1)

#calcualting the location

original\_distance = rand\_x \*\* 2 + rand\_y \*\* 2

#checking if the point lies inside the circle or square

if original\_distance <= 1:

points\_inside\_circle += 1

else:

points\_inside\_square += 1

#calculating the probabilty of the PI & returning

pi = 4 \* (points\_inside\_circle / iterations)

return pi

#main func to handle the userinput through argparse

def main():

#argument parser

parser = argparse.ArgumentParser(description='Monte Carlo simulation to estimate the value of pi.')

#command line arguments

parser.add\_argument('-i', '--iterations', type=int, help='Number of iterations')

parser.add\_argument('-j', '--json', help='Read iterations from a JSON file')

parser.add\_argument('-H', '--custom-help', action='store\_true', help='Display help message')

args = parser.parse\_args()

#the message for help when user enters '-h'

if args.custom\_help:

print("Monte Carlo Pi Estimation Tool")

print("Usage: python monte\_carlo\_pi.py -i <iterations>")

print("Options:")

print(" -i, --iterations Number of iterations")

print(" -j, --json Read iterations from a JSON file")

print(" -H, --custom-help Display this help message")

return

#if user chooses json file to enter iterations

if args.json:

try:

#opens the json file and reads the command

with open(args.json, 'r') as json\_file:

data = json.load(json\_file)

if 'iterations' in data:

#if 'iterations' is mentioned in file, it fetches the value

args.iterations = data['iterations']

else:

print("JSON file should contain 'iterations' key.")

return

#else for any problem displays erorr

except FileNotFoundError:

print(f"File not found: {args.json}")

return

#if user enters no iteration value in '-i' option

#or the json file doesnt contain 'iteration' value

#this message is displayed

if args.iterations is None:

print("You must specify the number of iterations either with -i or -j.")

return

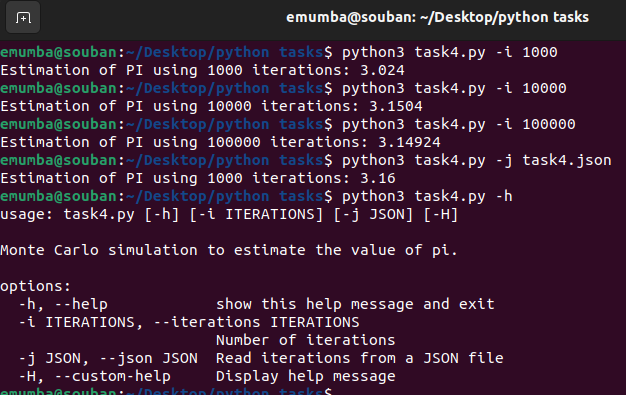
#estimation of pi value

pi = estimate\_pi(args.iterations)

print(f"Estimation of PI using {args.iterations} iterations:", pi)

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

main()



Task # 5

**Question**

You need to create 2 functions namely **square(list)** and **cube(list),** which takes the list as an argument and returns the list with all elements square and cube, respectively. Then pass the list of 1st 10 natural numbers(X = [1, 2, 3 , 4, 5, 6, 7, 8, 9, 10]) to each function and plot the square and cube graphs/plots using matplotlib **pyplot** module.

Note: If you have extra time, try labeling axis and give a suitable title to plots.

**Solution**

import matplotlib.pyplot as plt

#function for squaring the list entered

def square(list):

list\_of\_numbers2 = [x \*\* 2 for x in list]

return list\_of\_numbers2

#function for cubing the list entered

def cube(list):

list\_of\_numbers3 = [x \*\* 3 for x in list]

return list\_of\_numbers3

#declared list

list = []

print("Enter 10 numbers as a list")

#taking user input

for i in range(10):

number = int(input())

list.append(number)

#plotting function

def plotting\_func(list, Squared, Cubed):

#creating figure window of 10x5

plt.figure(figsize=(10, 5))

#subplot for squared number graph

plt.subplot(1,2,1)

#passing all the parameters for the plot

plt.plot(list,Squared,marker='o',linestyle='-', color = 'blue')

plt.title('Squared numbers')

plt.xlabel('Numbers')

plt.ylabel('Square')

#subplot for squared number graph

plt.subplot(1,2,2)

#passing all the parameters for the plot

plt.plot(list,Cubed,marker='o',linestyle='-', color = 'red')

plt.title('Cubed numbers')

plt.xlabel('Numbers')

plt.ylabel('Cube')

#to prevent overlap, it was overlapping

plt.tight\_layout()

#command to display the plots

plt.show()

#calling squared func

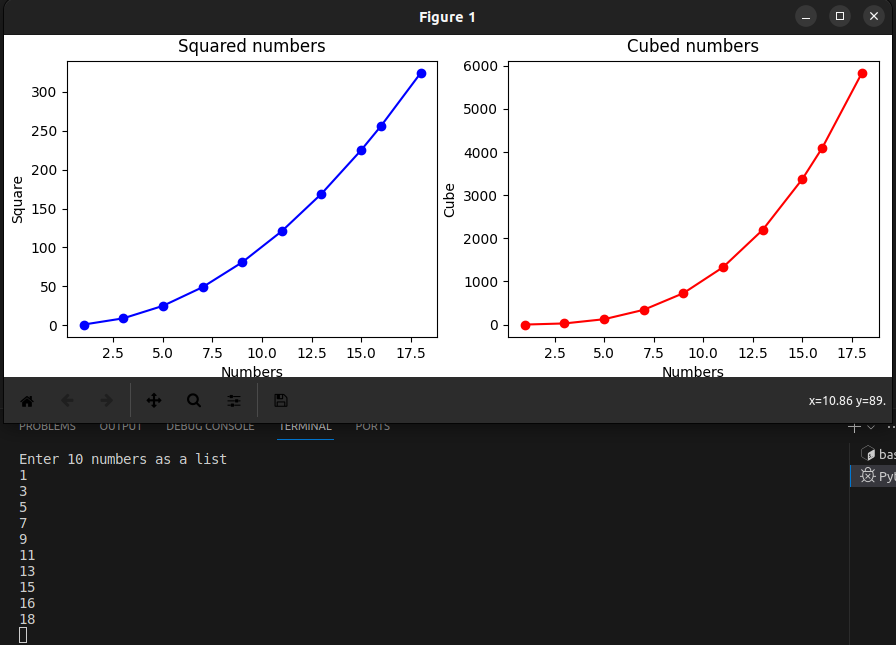
Squared = square(list)

#calling cube func

Cubed = cube(list)

#calling plotting func

plotting\_func(list,Squared,Cubed)



# Task # 6

**Question**

Make two python scripts **find\_divisibles.py** and **find\_divisibles\_async.py.** In the [*main*](https://www.geeksforgeeks.org/python-main-function/)driver codes of **find\_divisibles.py** call **find\_divisibles()** three times and in [*main*](https://www.geeksforgeeks.org/python-main-function/)driver code of **find\_divisibles\_async.py**, create three async tasks **async\_find\_divisibles().** Use the following arguments for both function calls:

(50800000, 34113)

(100052, 3210)

(500, 3)

Function **find\_divisibles(in\_range, divisor)** takes two arguments. The function should return a list of values, in range 1 to *in\_range,* which are divisible by the *divisor*. A log should be at the start of the function and at the end of the function. Both logs should include arguments (in\_range and divisor) to uniquely identify the function call. In the ending log, include the time this function call took from the start of the script (script start time - function end time).

For example:

find\_divisibles(20, 4):

Log( “find\_divisibles called with range 20 and divisor 4”)

//code

Log( “find\_divisibles ended with range 20 and divisor 4. It took x seconds”)

return [4,8,12,16,20].

*Async* function **async\_find\_divisibles(number\_range, divisor)** has the same functionality as **find\_divisibles(number\_range, divisor).** This function should be able to switch *asyncio context* whenever a divisible number is found.

For both scripts keep these points in mind:

* In the end [pprint](https://www.geeksforgeeks.org/pprint-data-pretty-printer-python/) the list returned by the second and third function call.
* Make sure to call functions in the given order.
* Before starting, make a virtual environment and install *asyncio* in that. Run these scripts in that venv.

Also, try with other arguments and observe how both functions differ.

**Solution**

**Find\_divisbles.py**

import time

def find\_divisibles(in\_range,divisor):

#fetching the time-1

start = time.time()

print("find\_divisibles called with range", in\_range, "and divisor",divisor,"\n")

#list variable for storing the number which are divisible by the divisor

numbers\_divisible\_by\_divisor= []

#iterative loop to check dividend and appending the

#required number in the list

for num in range (1, in\_range+1):

if num % divisor == 0:

numbers\_divisible\_by\_divisor.append(num)

print(numbers\_divisible\_by\_divisor)

# fetching the time-2, to calculate the difference

end = time.time()

#finding the elapsed time by difference

total\_time\_elapsed = end - start

print("find\_divisibles called with range" , in\_range , "and divisor" , divisor, "and it took" , total\_time\_elapsed , "second \n")

return(numbers\_divisible\_by\_divisor)

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

# function calling

print("\n")

find\_divisibles(5080000,34113)

print("\n")

find\_divisibles(100052,3210)

print("\n")

find\_divisibles(500,3)

**Find\_divisbles\_async.py**

import asyncio

import time

from pprint import pprint

async def async\_find\_divisibles(in\_range,divisor):

#fetching the time-1

start = time.time()

print("find\_divisibles called with range", in\_range, "and divisor",divisor,"\n")

#list variable for storing the number which are divisible by the divisor

numbers\_divisible\_by\_divisor= []

#iterative loop to check dividend and appending the

#required number in the list

for num in range (1, in\_range+1):

if num % divisor == 0:

numbers\_divisible\_by\_divisor.append(num)

await asyncio.sleep(0)

print(numbers\_divisible\_by\_divisor)

# fetching the time-2, to calculate the difference

end = time.time()

#finding the elapsed time by difference

total\_time\_elapsed = end - start

print("find\_divisibles called with range" , in\_range , "and divisor" , divisor, "and it took" , total\_time\_elapsed , "second \n")

return(numbers\_divisible\_by\_divisor)

# This is the main co-routine that calls the async\_find\_divisibles function

async def main():

# function calling

print("\n")

result1 = await async\_find\_divisibles(5080000,34113)

print("\n")

result2 = await async\_find\_divisibles(100052,3210)

print("\n")

result3 = await async\_find\_divisibles(500,3)

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

asyncio.run(main())

**Commands to run it through terminal**

**To install python3.10-venv**

sudo apt install python3.10-venv

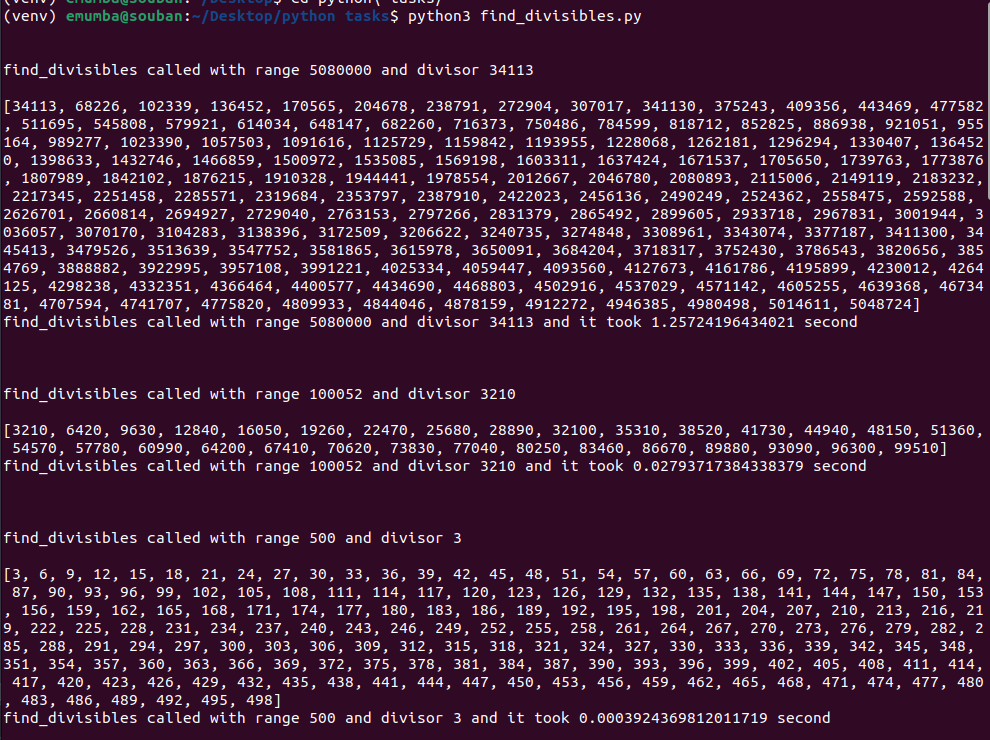
**To create the virtual environment**

python3.10 -m venv venv

**To activate the virtual environment**

source venv/bin/activate

****

****

# 

# 

# 

# 

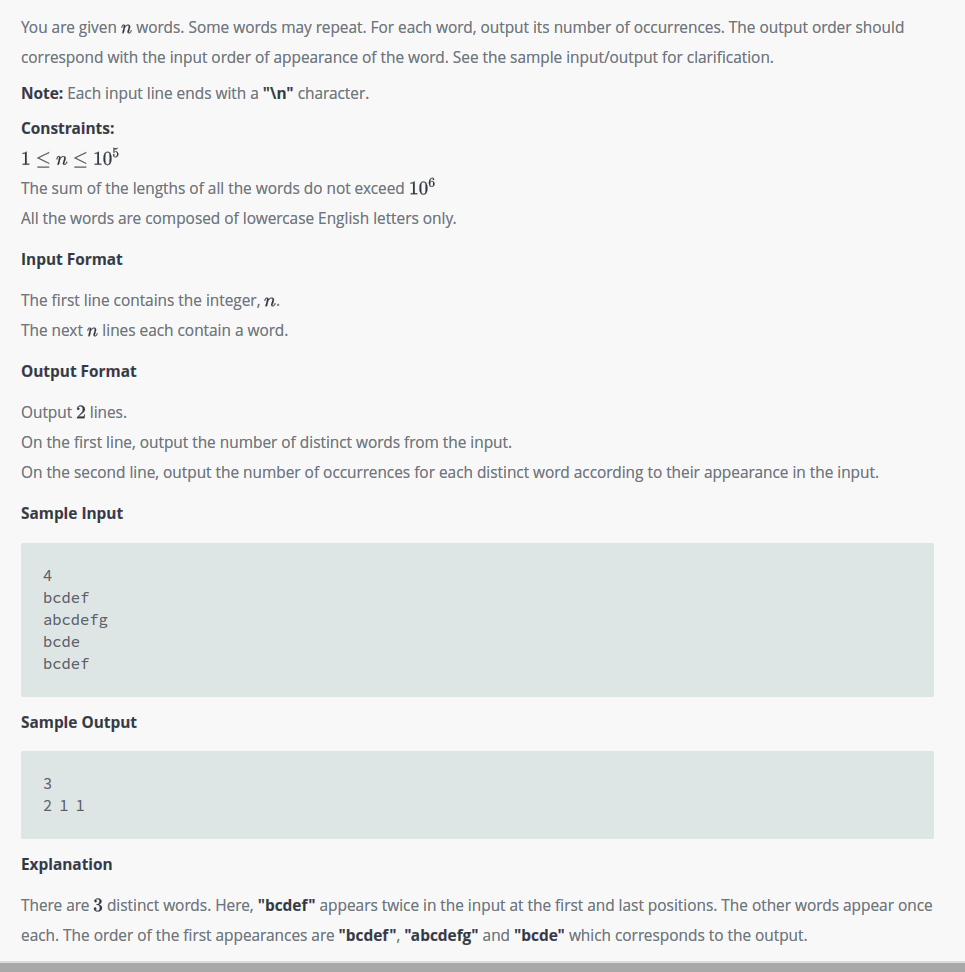
# 

# 

# 

# Task # 9

**Question**



**Solution**

print("How many words do you want to enter? ")

#taking input from user about the amount of number of words

n\_number\_of\_words = int(input())

#initialize an empty dictionary to store words

dictionary = {}

#an empty list to store all type of words

list\_of\_words = []

#for storing the count of distinct words

distinct\_words = []

#loop to process each words

for i in range(n\_number\_of\_words):

temp\_word = input() #reading input from users

if temp\_word not in dictionary: #if the word is not present in the dictionary

list\_of\_words.append(temp\_word) #append that words inside the dictionary

dictionary[temp\_word] =1 #change count into 1 for that specific word

#if the word is present

else:

dictionary[temp\_word] +=1 #increment the count for the existing words

#loop for generating the count of distinct words inside the dictionary

for temp\_word in list\_of\_words:

distinct\_words.append(str(dictionary[temp\_word]))

#display the number of distinct words

print(len(list\_of\_words))

#printing the occurences of each words without spaces

print("".join(distinct\_words))

