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]

```
emumba@emumba-Inspiron-5570:~/Desktop/python tasks$  cd /home/emumba/Desktop/python\ tasks ; /usr/bin/env /bin/python3 /home/emumba/.vscode/extensions/ms-python.python-2023.16.0/pythonFi
les/lib/python/debugpy/adapter/../../debugpy/launcher 47817 -- /home/emumba/Desktop/python\ tasks/task1.py
*This program will print only the numbers between 2000-3500 which are divisible by 7 but not multiple of 5*
2002,2009,2016,2023,2037,2044,2051,2058,2072,2079,2086,2093,2107,2114,2121,2128,2142,2149,2156,2163,2177,2184,2191,2198,2212,2219,2226,2233,2247,2254,2261,2268,2282,2289,2296,2303,2317,2
324,2331,2338,2352,2359,2366,2373,2387,2394,2401,2408,2422,2429,2436,2443,2457,2464,2471,2478,2492,2499,2506,2513,2527,2534,2541,2548,2562,2569,2576,2583,2597,2604,2611,2618,2632,2639,26
46,2653,2667,2674,2681,2688,2702,2709,2716,2723,2737,2744,2751,2758,2772,2779,2786,2793,2807,2814,2821,2828,2842,2849,2856,2863,2877,2884,2891,2898,2912,2919,2926,2933,2947,2954,2961,296
8,2982,2989,2996,3003,3017,3024,3031,3038,3052,3059,3066,3073,3087,3094,3101,3108,3122,3129,3136,3143,3157,3164,3171,3178,3192,3199,emumba@emumba-Inspiron-5570:~/Desktop/python tasks$
```

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()
```

```
emumba@emumba-Inspiron-5570:~/Desktop/python tasks$ python3 task2.py 5 3 3 2 --help
usage: task2.py [-h] int1 int2 int3 int4

Enter integer or floating-point values for the directions mentioned:

positional arguments:
  int1
  int2
  int3
  int4

options:
  -h, --help  show this help message and exit
emumba@emumba-Inspiron-5570:~/Desktop/python tasks$ python3 task2.py 5 3 3 2
UP                                5
DOWN                              3
LEFT                              3
RIGHT                             2
T. Travelled distance             13
distance from original position   2
```

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}")
```

```
emumba@souban: ~/Desktop/python tasks
emumba@souban:~$ cd Desktop/python\ tasks/
emumba@souban:~/Desktop/python tasks$ python3 task3.py
Byte Order:                         Little Endian

Core(s) per socket:                 4

Socket(s):                          1

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

CPU max MHz:                        4000.0000
CPU min MHz:                        400.0000

Virtualization:                     VT-x

L1d cache:                          128 KiB (4 instances)
L1i cache:                          128 KiB (4 instances)

L2 cache:                           1 MiB (4 instances)

L3 cache:                           8 MiB (1 instance)

RAM Memory:                          31Gi

Summary.txt file created at: /home/emumba/Details
emumba@souban:~/Desktop/python tasks$
```

```
emumba@souban:~/Desktop/python tasks$  cd /home/emumba/Desktop/python\ tasks ; /usr/bin/env /
on3 /home/emumba/.vscode/extensions/ms-python.python-2023.16.0/pythonFiles/lib/python/debugp
/../../debugpy/launcher 59731 -- /home/emumba/Desktop/python\ tasks/task3.py
Byte Order:                         Little Endian

Core(s) per socket:                 4

Socket(s):                          1

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

CPU max MHz:                        4000.0000
CPU min MHz:                        400.0000

Virtualization:                     VT-x

L1d cache:                          128 KiB (4 instances)
L1i cache:                          128 KiB (4 instances)

L2 cache:                           1 MiB (4 instances)

L3 cache:                           8 MiB (1 instance)

RAM Memory:                          31Gi

Summary.txt file created at: /home/emumba/Details
emumba@souban:~/Desktop/python tasks$
```

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()
```

```
emumba@souban: ~/Desktop/python tasks
emumba@souban:~/Desktop/python tasks$ python3 task4.py -i 1000
Estimation of PI using 1000 iterations: 3.024
emumba@souban:~/Desktop/python tasks$ python3 task4.py -i 10000
Estimation of PI using 10000 iterations: 3.1504
emumba@souban:~/Desktop/python tasks$ python3 task4.py -i 100000
Estimation of PI using 100000 iterations: 3.14924
emumba@souban:~/Desktop/python tasks$ python3 task4.py -j task4.json
Estimation of PI using 1000 iterations: 3.16
emumba@souban:~/Desktop/python tasks$ python3 task4.py -h
usage: task4.py [-h] [-i ITERATIONS] [-j JSON] [-H]

Monte Carlo simulation to estimate the value of pi.

options:
  -h, --help            show this help message and exit
  -i ITERATIONS, --iterations ITERATIONS
                        Number of iterations
  -j JSON, --json JSON  Read iterations from a JSON file
  -H, --custom-help     Display help message
emumba@souban:~/Desktop/python tasks$
```

Task # 5

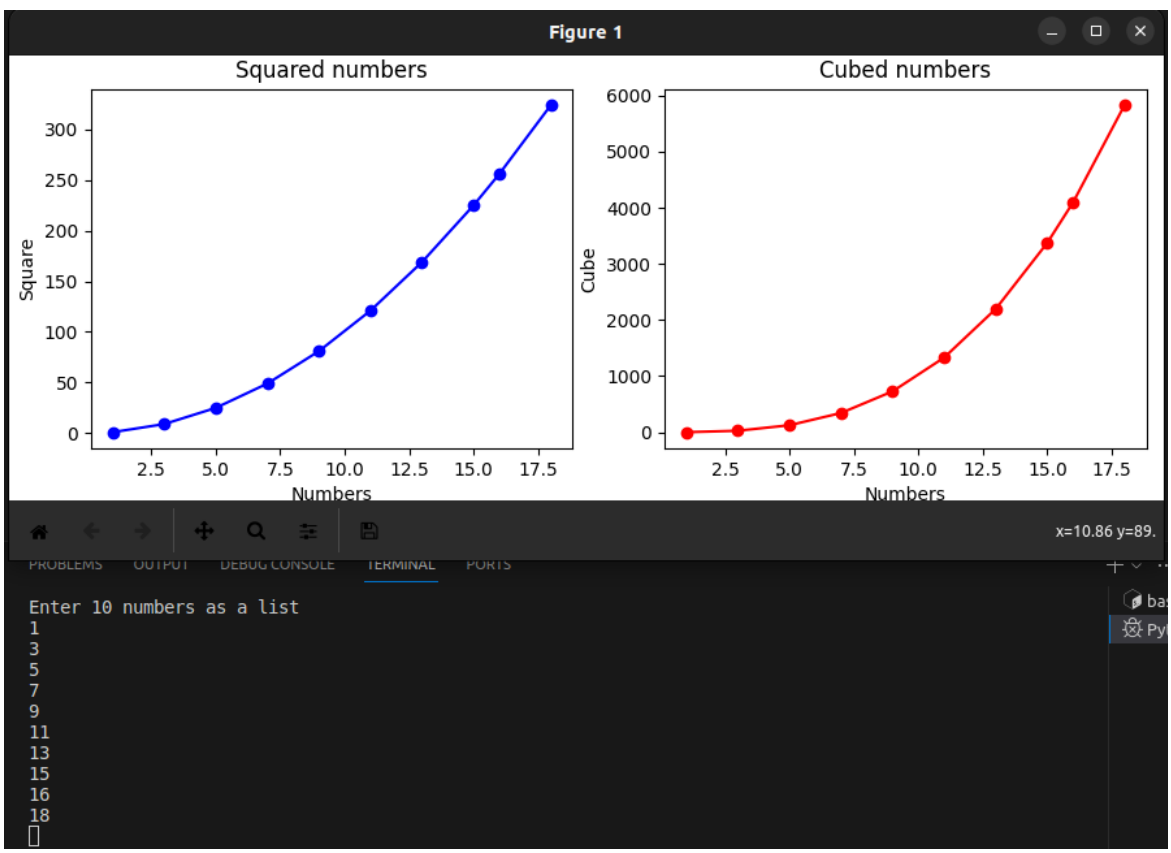
Question

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* driver codes of **find_divisibles.py** call **find_divisibles()** three times and in *main* 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 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

```
emumba@souban:~$ python3.10 -m venv venv
emumba@souban:~$ source venv/bin/activate
(venv) emumba@souban:~$ cd Desktop
```

```
(venv) emumba@souban:~/Desktop/python tasks$ python3 find_divisibles.py


find_divisibles called with range 5080000 and divisor 34113

[34113, 68226, 102339, 136452, 170565, 204678, 238791, 272904, 307017, 341130, 375243, 409356, 443469, 477582
, 511695, 545808, 579921, 614034, 648147, 682260, 716373, 750486, 784599, 818712, 852825, 886938, 921051, 955
164, 989277, 1023390, 1057503, 1091616, 1125729, 1159842, 1193955, 1228068, 1262181, 1296294, 1330407, 136452
0, 1398633, 1432746, 1466859, 1500972, 1535085, 1569198, 1603311, 1637424, 1671537, 1705650, 1739763, 1773876
, 1807989, 1842102, 1876215, 1910328, 1944441, 1978554, 2012667, 2046780, 2080893, 2115006, 2149119, 2183232,
 2217345, 2251458, 2285571, 2319684, 2353797, 2387910, 2422023, 2456136, 2490249, 2524362, 2558475, 2592588,
2626701, 2660814, 2694927, 2729040, 2763153, 2797266, 2831379, 2865492, 2899605, 2933718, 2967831, 3001944, 3
036057, 3070170, 3104283, 3138396, 3172509, 3206622, 3240735, 3274848, 3308961, 3343074, 3377187, 3411300, 34
45413, 3479526, 3513639, 3547752, 3581865, 3615978, 3650091, 3684204, 3718317, 3752430, 3786543, 3820656, 385
4769, 3888882, 3922995, 3957108, 3991221, 4025334, 4059447, 4093560, 4127673, 4161786, 4195899, 4230012, 4264
125, 4298238, 4332351, 4366464, 4400577, 4434690, 4468803, 4502916, 4537029, 4571142, 4605255, 4639368, 46734
81, 4707594, 4741707, 4775820, 4809933, 4844046, 4878159, 4912272, 4946385, 4980498, 5014611, 5048724]
find_divisibles called with range 5080000 and divisor 34113 and it took 1.25724196434021 second



find_divisibles called with range 100052 and divisor 3210

[3210, 6420, 9630, 12840, 16050, 19260, 22470, 25680, 28890, 32100, 35310, 38520, 41730, 44940, 48150, 51360,
 54570, 57780, 60990, 64200, 67410, 70620, 73830, 77040, 80250, 83460, 86670, 89880, 93090, 96300, 99510]
find_divisibles called with range 100052 and divisor 3210 and it took 0.02793717384338379 second



find_divisibles called with range 500 and divisor 3

[3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57, 60, 63, 66, 69, 72, 75, 78, 81, 84,
 87, 90, 93, 96, 99, 102, 105, 108, 111, 114, 117, 120, 123, 126, 129, 132, 135, 138, 141, 144, 147, 150, 153
, 156, 159, 162, 165, 168, 171, 174, 177, 180, 183, 186, 189, 192, 195, 198, 201, 204, 207, 210, 213, 216, 21
9, 222, 225, 228, 231, 234, 237, 240, 243, 246, 249, 252, 255, 258, 261, 264, 267, 270, 273, 276, 279, 282, 2
85, 288, 291, 294, 297, 300, 303, 306, 309, 312, 315, 318, 321, 324, 327, 330, 333, 336, 339, 342, 345, 348,
351, 354, 357, 360, 363, 366, 369, 372, 375, 378, 381, 384, 387, 390, 393, 396, 399, 402, 405, 408, 411, 414,
 417, 420, 423, 426, 429, 432, 435, 438, 441, 444, 447, 450, 453, 456, 459, 462, 465, 468, 471, 474, 477, 480
, 483, 486, 489, 492, 495, 498]
find_divisibles called with range 500 and divisor 3 and it took 0.0003924369812011719 second
```

emumba@souban: ~/Desktop/python tasks

```
emumba@souban:~$ cd Desktop/python\ tasks/
emumba@souban:~/Desktop/python tasks$ python3.10 -m venv myvenv
emumba@souban:~/Desktop/python tasks$ source venv/bin/activate
bash: venv/bin/activate: No such file or directory
emumba@souban:~/Desktop/python tasks$ source myvenv/bin/activate
(myvenv) emumba@souban:~/Desktop/python tasks$ python3.10 find_divisibles_async.py


find_divisibles called with range 5080000 and divisor 34113

[34113, 68226, 102339, 136452, 170565, 204678, 238791, 272904, 307017, 341130, 375243, 409356, 443469, 477582, 511695, 545808, 579
921, 614034, 648147, 682260, 716373, 750486, 784599, 818712, 852825, 886938, 921051, 955164, 989277, 1023390, 1057503, 1091616, 11
25729, 1159842, 1193955, 1228068, 1262181, 1296294, 1330407, 1364520, 1398633, 1432746, 1466859, 1500972, 1535085, 1569198, 160331
1, 1637424, 1671537, 1705650, 1739763, 1773876, 1807989, 1842102, 1876215, 1910328, 1944441, 1978554, 2012667, 2046780, 2080893, 2
115006, 2149119, 2183232, 2217345, 2251458, 2285571, 2319684, 2353797, 2387910, 2422023, 2456136, 2490249, 2524362, 2558475, 25925
88, 2626701, 2660814, 2694927, 2729040, 2763153, 2797266, 2831379, 2865492, 2899605, 2933718, 2967831, 3001944, 3036057, 3070170,
3104283, 3138396, 3172509, 3206622, 3240735, 3274848, 3308961, 3343074, 3377187, 3411300, 3445413, 3479526, 3513639, 3547752, 3581
865, 3615978, 3650091, 3684204, 3718317, 3752430, 3786543, 3820656, 3854769, 3888882, 3922995, 3957108, 3991221, 4025334, 4059447,
 4093560, 4127673, 4161786, 4195899, 4230012, 4264125, 4298238, 4332351, 4366464, 4400577, 4434690, 4468803, 4502916, 4537029, 457
1142, 4605255, 4639368, 4673481, 4707594, 4741707, 4775820, 4809933, 4844046, 4878159, 4912272, 4946385, 4980498, 5014611, 5048724
]
find_divisibles called with range 5080000 and divisor 34113 and it took 1.2003183364868164 second



find_divisibles called with range 100052 and divisor 3210

[3210, 6420, 9630, 12840, 16050, 19260, 22470, 25680, 28890, 32100, 35310, 38520, 41730, 44940, 48150, 51360, 54570, 57780, 60990,
 64200, 67410, 70620, 73830, 77040, 80250, 83460, 86670, 89880, 93090, 96300, 99510]
find_divisibles called with range 100052 and divisor 3210 and it took 0.023482799530029297 second



find_divisibles called with range 500 and divisor 3

[3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57, 60, 63, 66, 69, 72, 75, 78, 81, 84, 87, 90, 93, 96, 99,
102, 105, 108, 111, 114, 117, 120, 123, 126, 129, 132, 135, 138, 141, 144, 147, 150, 153, 156, 159, 162, 165, 168, 171, 174, 177,
180, 183, 186, 189, 192, 195, 198, 201, 204, 207, 210, 213, 216, 219, 222, 225, 228, 231, 234, 237, 240, 243, 246, 249, 252, 255,
258, 261, 264, 267, 270, 273, 276, 279, 282, 285, 288, 291, 294, 297, 300, 303, 306, 309, 312, 315, 318, 321, 324, 327, 330, 333,
336, 339, 342, 345, 348, 351, 354, 357, 360, 363, 366, 369, 372, 375, 378, 381, 384, 387, 390, 393, 396, 399, 402, 405, 408, 411,
414, 417, 420, 423, 426, 429, 432, 435, 438, 441, 444, 447, 450, 453, 456, 459, 462, 465, 468, 471, 474, 477, 480, 483, 486, 489,
492, 495, 498]
find_divisibles called with range 500 and divisor 3 and it took 0.003137350082397461 second
```

Task # 9

Question

You are given n words. Some words may repeat. For each word, output its number of occurrences. The output order should correspond with the input order of appearance of the word. See the sample input/output for clarification.

Note: Each input line ends with a **"\n"** character.

Constraints:

$1 \leq n \leq 10^5$

The sum of the lengths of all the words do not exceed 10^6

All the words are composed of lowercase English letters only.

Input Format

The first line contains the integer, n.

The next n lines each contain a word.

Output Format

Output 2 lines.

On the first line, output the number of distinct words from the input.

On the second line, output the number of occurrences for each distinct word according to their appearance in the input.

Sample Input

```
4
bcdef
abcdefg
bcde
bcdef
```

Sample Output

```
3
2 1 1
```

Explanation

There are 3 distinct words. Here, **"bcdef"** appears twice in the input at the first and last positions. The other words appear once each. The order of the first appearances are **"bcdef"**, **"abcdefg"** and **"bcde"** which corresponds to the output.

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))
```

```
How much words you want to enter?
4
bcdef
abcdef
bcde
bcdef
3
211
```