Getting started

Read a pixel from the top left corner of a given image and write its red, green and blue components to the console output.

Input

Path to the image is given through standard input. Please refer to the solution below for more details.

Output

Result should be written to the console output in three lines, using the following format:

Red: <red\_component>

Green: <green\_component>

Blue: <blue\_component>

Please refer to the solution below for more details.

Example

For a completely white image, the expected output is:

Red: 255

Green: 255

Blue: 255

Scoring

Correctly solving this task brings you 1 point. You can see the solution below.

Solution

Feel free to copy/paste the given solution to see how the grading process works.

Python

import numpy as np

from PIL import Image

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

image\_path = input()

image\_file = Image.open(image\_path)

image = np.array(image\_file)

pixel = image[0, 0]

print("Red: {0}".format(pixel[0]))

print("Green: {0}".format(pixel[1]))

print("Blue: {0}".format(pixel[2]))

Checking the status of a submitted solution

You can find more details about your submitted solutions in the *Overview* tab to the left.

There are two groups of test cases (data sets):

* *Public data set* is used for developing your solution. After you submit your solution, you will be able to see how well your solution performs against this data set. *Public data set* is not used for calculating the final score. Public data set is available during the homework.
* *Private data set* is used for testing your solution. The final score will be measured against this data set. *Private data set* and the final score will be available after the homework finishes. *Private data set* contains different data than the *public data set*, but the type of data (e.g. text length, number of files, image size...) is roughly the same.

When you click on more details for a submitted solution, you can see the status, score and time and memory the solution uses per test case. Private test cases are marked with *?* during the homework.

Rock My World

Description:

You are a member of an upcoming indie music band that is trying to make it big in the world, but you have one small problem - your music sucks. That's why you will have to to use the power of data wrangling to help you. Right at your fingertips you have a lot of badly formatted json files which were scrapped together from various sources. If you manage to clean them up and get some insights from them you just might get your band's name out there.

Json files are given in one of the following file hierarchy structures:

{dataset}/{year}/{month}/{day}/{country}/{city}/{venue}.json

{dataset}/{year\_month\_day}/{country}/{city}/{venue}.json

{dataset}/{country}/{city}/{year\_month\_day}/{venue}.json

The naming scheme is not standardized, we will have to clean this up during our analysis (there is no need to rename the files in the input):

* years are given in two formats: either full year, 1995, or just the last two digits 95. The data we have is from interval 1960 to 2022.
* months and days might have trailing zeroes at the beginning
* country, city and venue names might have underscore or dash as the word delimiter - transform all these to underscore for uniformity
* country names might have the word 'the' included as a prefix (i.e. 'the\_slovenia') - delete these prefixes if they exist

The {venue}.json files contain a list of all concerts in the venue for the given location in the given date. The location (country and city) and the date (year, month, day) can be extracted from the full path of the file.

In the json files each concert has the following fields:

* "**band\_name**": name of the band
* "**is\_indie**": boolean value which indicates whether the band is indie or not
* "**attendance**": total number of people that attended the concert

Additionally the json files themselves are problematic

* some .json files are empty - they can be discarded
* some files use standard json format while some use line-delimited json format, both should be analysed
* some concerts have missing data fields
  + if "band\_name" field is missing the whole concert should be discarded
  + if "is\_indie" field is missing the value false should be imputed
  + if "attendance" field is missing the average attendance from all concerts in the venue should be imputed, if there are no such concerts the average attendance from all concerts from all venues should be imputed

A) **Calculate the total number of json files (including invalid and empty json files)**

B) **Calculate the number of countries that have at least one valid concert.**

C) **Calculate the city with most concerts.** (If there are multiple cities with most concerts sort them alphabetically in ascending order and print the first one. For example if we have(belgrade, chicago, denver) then you should print belgrade)

D) **Calculate the best 3 cities by the amount of people who went on indie concerts.**

E) **Calculate the 3 most popular bands by the average amount of people who attended their concerts. Only take into consideration concerts for which there was at least one indie artist performing in the same venue at the same day.**

With this info we should be able to piggy-back on the fame of popular indie-friendly artists and choose the best city to do so!

Additional information:

* there can be venues with the same name in different locations
* bands can be at venues on different location at the same date

Input format:

Your program should take in path to the dataset through standard input.

Output format:

Output answers to subtasks in 5 separate lines for each of the subtasks in order. If you don't have the answer to some answer leave a blank line.

* A: integer, total number of json files
* B: integer, number of countries that have at least one valid concert
* C: string, city with most concerts
* D: string, comma separated list of 3 cities chosen as explained in subtask D
* E: string, comma separated list of 3 bands chosen as explained in subtask E

Example:

Input:

Important: you are given only path to the dataset (root\_path) through standard input, bellow you can see how the whole directory structure and files look like.

File1 Path:

{root\_path}/1999/12/3/united\_states\_of\_america/houston/thirtytwo.json

File1 Content:

[

{

"attendance": 10000,

"band\_name": "Pink Floyd",

"is\_indie": true

},

{

"is\_indie": false,

"attendance": 90000,

"band\_name": "Aqua"

},

{

"band\_name": "Michael Jackson",

"attendance": 30000,

"is\_indie": false

}

]

File2 Path:

{root\_path}/1999\_12\_04/the\_united\_states\_of\_america/new\_york/groove.json

File2 Content:

[

{

"band\_name": "Pink Floyd",

"is\_indie": true,

"attendance": 15000

}

]

File3 Path:

{root\_path}/1999/12/05/united\_states\_of\_america/los\_angeles/hitmeup.json

File3 Content:

{ "band\_name": "Michael Jackson", "attendance": 1000, "is\_indie": false }

{ "band\_name": "Ceca", "attendance": 1000, "is\_indie": false }

File4 Path:

{root\_path}/1999/12/6/united\_states\_of\_america/chicago/icrywhenifly.json

File4 Content:

[

{

"is\_indie": true,

"band\_name": "Pink Floyd",

"attendance": 5000

}

]

File5 Path:

{root\_path}/1999/12/7/the\_united\_states\_of\_america/chicago/icrywhenifly.json

File5 Content:

Output:

5

1

houston

los\_angeles,new\_york,houston

Aqua,Michael Jackson,Pink Floyd

Explanation:

* there is a total of 5 json files
* there is 1 country in which there were concerts
* houston had 3 most concerts while the other cities had only 1
* new\_york had total attendance of 15000 on indie bands, houston had 10000 and chicago had 5000
* Aqua had 90000 average attendance on venues where indie artists were performing at the same day, Michael Jackson had 20000 (the concerts from File3 are not relevant because there was no indie artist in that vanue at that day) and Pink Floyd had (10000+15000+5000)/3 = 10000

Limitations

Time limit for task execution is 3 seconds per test case and the size limit is 128MB.

Allowed python packages:

1. python standard library
2. numpy
3. pandas

Scoring

* Correct result for subtask 1 brings 10 of points per test case.
* Correct result for subtask 2 brings 20 of points per test case.
* Correct result for subtask 3 brings 20 of points per test case.
* Correct result for subtask 4 brings 25 of points per test case.
* Correct result for subtask 5 brings 25 of points per test case.

Datasets

There are two data sets:

* *Public data set* is used for developing your solution. After you submit your solution, you will be able to see how well your solution performs against this data set. *Public data set* is not used for calculating the final score. Public data set is available [**here**](https://petljamediastorage.blob.core.windows.net/psiml/2022/rockmyworld-public.zip).
* *Private data set* is used for testing your solution. The final score will be measured against this data set. *Private data set* and the final score will be available after the homework finishes. *Private data set* contains different data than the *public data set*, but the type of data is roughly the same (the same constraints apply).

Sudoku

Description

Your little cousin has just learned how to solve a sudoku. He finds the game interesting but is still not good at it, so he often sends you an image asking for your help. You don’t always have time to help him, so you want to automatize this task.

Task 1: Read the table

You are given an image of a standard 9x9 sudoku table. You already have the images of your cousin’s usual handwriting, and you can assume he writes all 9 digits fairly consistently, but you know you can encounter several problems:

1. He is taking the pictures carefully, so the table is perfectly aligned, but you can expect the image you receive to be rotated by 180 degrees, or by 90 degrees in either direction.
2. The sudoku table is not the same size each time, and the borders can be of slightly different widths.
3. The table can be in different parts of the image.
4. Although the digits are always centered, they can vary in size and take up different portions of the tiles.

Knowing all of this, you want to recognize all digits in the sudoku table, as well as the positions of the empty tiles.

Additional notes:

* All images are in PNG format.
* Each tile is a square with a side of at least 32 and up to 64 pixels.
* The sudoku table is always white, and your cousin colors the rest of the image in black before sending it.
* He is always using a black marker for writing.

Task 2: Check the solution

You want to help him finish his game. You can assume that most of the numbers are already written, with up to 11 empty tiles. Find the numbers to enter in the empty tiles to make solved sudoku.

Note: In the solved sudoku no rows, columns, or sub-boxes contain the same digit.

Input format

The input is a path to a folder, provided as a command-line parameter. The input folder contains another folder and one png image. The png is the image of unsolved sudoku and the subfolder contains the sample digits written in your cousin’s handwriting.

Output format

The resulting table should be printed to the standard output, one row per line. Each line should be printed in csv format (as 9 comma-separated digits). Empty tiles should be represented with a zero.

The following 9 lines should contain the solved sudoku, in the same format. Each value 0 from the previous 9 lines should be replaced with a correct digit.

Scoring

Each test case brings a maximum of 100 points. Full correctly detected sudoku table brings 50 points per test case. The correct sudoku solution brings additional 50 points only if there are no errors in the previous output.

Datasets

There are two data sets:

Public data set is used for developing your solution. After you submit your solution, you will be able to see how well your solution performs against this data set. Public data set is not used for calculating the final score. Public data set is available [**here**](https://petljamediastorage.blob.core.windows.net/psiml/2022/sudoku-public.zip).

Private data set is used for testing your solution. The final score will be measured against this data set. Private data set and the final score will be available after the homework finishes. Private data set contains different data than the public data set, but the type of data is roughly the same (the same constraints apply).

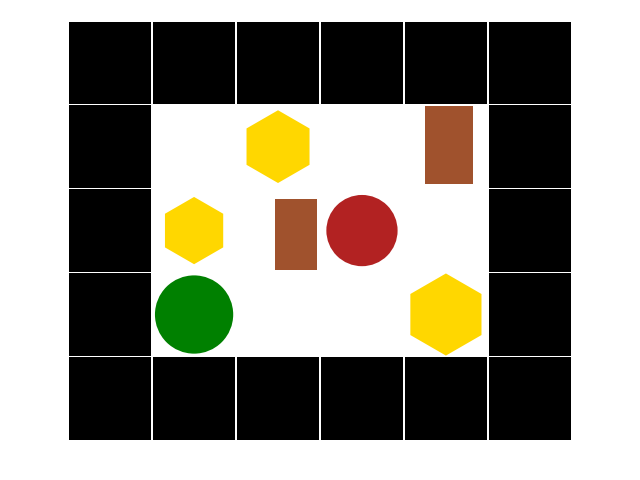
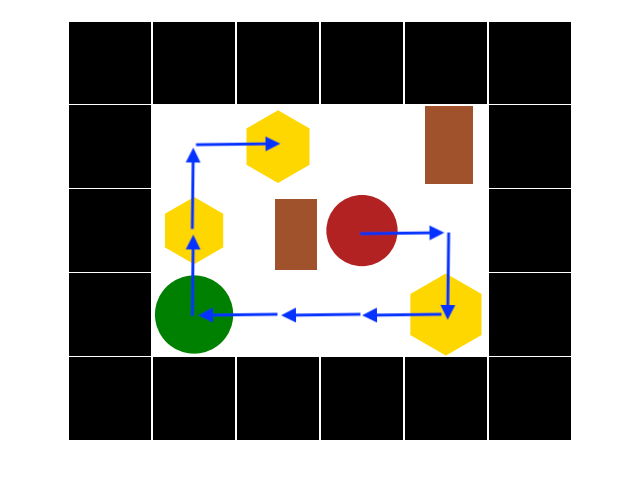
Limitations

The time limit for task execution is 1 second per input image.

Allowed python packages: numpy PIL all packages from the standard python library

Roomba

You went on a summer vacation and left a Roomba to clean your room while you are away. However, you forgot to calibrate it for the room (it has no idea where it is, where walls, obstacles and dirt are, nor where the charging station is).

Task 1: Calibration (10%)

You can access images from the roomba's station sensory array that provides a top-down representation of the room as a grid of square tiles. Tiles can be either empty or contain exatly one object (se the objects below) The grid tiles are colored according to the following scheme:

1. white: empty spaces,
2. black: walls (impassable obstacle).

Additionally, tiles can contain other objects of various sizes and positions inside the tile

1. red circle: roomba,
2. green circle: charging station,
3. brown rectangle: furniture (impassable obstacle),
4. yellow hexagon: dust and dirt that you need to clean.

Your first task is to process this image and provide your roomba with a "state of the world". Most importantly, you need to provide the following list of integers: [H, W] where:

* H: height of the room including walls,
* W: width of the room including walls, Then you need to output the room\_matrix in the following format (the example is for a 4x3 room):

[v00, v01, v02]

[v10, v11, v12]

[v20, v21, v22]

[v30. v31. v32]

where the value vij corresponds to the value of i-th row and j-th column according to the following legend: Legend:

* 'empty': 0
* 'wall': 1
* 'roomba': 2
* 'station': 3
* 'furniture': 4
* 'dirt': 5

Task 2: Path-Finding (10%)

It is of utmost importance that the roomba can find and reach the charging station before its battery runs out. Roomba gets charged when on the same tile as the charging station. Your second task is to provide the list: [A1, A2,...As] where

* A1...As - a list of movement actions chosen from {'u', 'd', 'l', 'r'} (meaning 'up', 'down', 'left', 'right' respectively) that results in roomba reaching the charging station in the minimum number of steps.

Task 3: Basic Cleaning (20%)

Next, roomba needs to find and clean dirt. Your third task is to provide the list: [A1, A2,...Ad] where

* A1...Ad - a list of movement actions chosen from {'u', 'd', 'l', 'r'} (meaning 'up', 'down', 'left', 'right' respectively) that results in roomba cleaning the room (cleaning all the dust and dirt piles) in the minimum number of steps.

A tile with dust or dirt is considered cleaned if at any point roomba was on that tile.

Task 4: Efficient Cleaning (40%)

Your roomba has a battery capacity Cmax measured in number of actions (one of 'up', 'down', 'left', 'right') it can take before needing to recharge. Recharging is done simply by stepping onto the tile with the charging station at which point roomba's battery gets maximum charges Cmax. When you first ping your roomba it will have C charges left (0 < C <= Cmax). You don't want to waste the electricity, so your fourth task is to provide the list: [A1, A2,...Ae] where

* A1...Ae - a list of movement actions chosen from {'u', 'd', 'l', 'r'} (meaning 'up', 'down', 'left', 'right' respectively) that results in roomba cleaning the room (cleaning all the dust and dirt piles) in the minimum number of steps while never running out of charges.

Task 5: Perfect Cleaning (20%)

Holiday sights and spirits fuel your fantasy, so you dream big: "when you come back, you're going to put a Tesla batery in your roomba and stop caring about the charges". Also, you are a perfectionist and you don't want to rely on the built-in dust and dirt sensors. You want every part of your room as clean as possible so you figure that there might be a way to clean the whole room by cleaning each tile exactly once; if not, you still want to clean each tile while minimizing the number of movement actions. Your fifth task is to provide the list [Ot, A1...Ap] where:

* Ot - number of over-cleaned tiles (tiles you had to visit more than once in order to clean every tile at least once); ideally this should be zero;
* A1...Ap - the exact list of the minimum number of movement actions that results in perfect cleaning.

Note: Charging station tile should be treated as an empty tile.

Additional Notes and Limits

1. All images are in PNG format
2. Room will always be rectangular with square tiles and:
   * minimum size 4x4 tiles
   * maximum size 8x8 tiles
3. There will always be:
   * exactly one roomba,
   * exactly one charging station,
   * at least one and at most sixteen dust/dirt piles,
   * at most sixteen tiles with furniture.
4. Orientation
   * (0,0) is the bottom left corner of the room
   * X increases from down to up
   * Y increases from left to right
5. The room will always be solvable, i.e. there will always be a path between the roomba and the station and the dust/dirt piles that is not blocked.

Input

inputImagePath (provided as command line parameter).

Output

Your program should output to stdout using the print() function:

# some code

print([C, Cmax])

# some code

print('Task 1')

print([H, W])

print(room\_matrix)

# some code

print('Task 2')

print(task2\_list)

# some code

print('Task 3')

print(task3\_list)

# some code

print('Task 4')

print(task4\_list)

# some code

print('Task 5')

print(task5\_list)

Note: Your output needs to be in the same format as seen in the output examples of the public dataset.

Examples

You can find example inputs and outputs in datasets\roomba\public\input and datasets\roomba\public\output respectively. Download the dataset [**here**](https://petljamediastorage.blob.core.windows.net/psiml/2022/roomba-public.zip).

Calling

Your programm will be called as a following command: Python3 roomba.py inputImagePath C, Cmax

Limitations

Time limit for task execution is TODO seconds per input image, on a TODO machine.

Allowed python packages:

1. pillow
2. numpy