

*Figure 1*

Foundation Certificate for Higher Education

Module: DOC334 Computer Programming

Module Leader: Mr. Nishan Saliya

Assignment Type: Individual

Submission Date: 2024.03.30

Student ID: 20231264

Student Name: Ranuga Disansa Belpa Gamage

Student Email: ranuga.20231264@iit.ac.lk

Acknowledgment

I want to express my appreciation to those who have helped complete this report.

I have been able to complete this report because of the Academic advisors Mr. Nishan Saliya and Ms. Shafka Fuard for their invaluable support, mentorship, and feedback, and the faculty members of DOC 334 (Computer Programming).

Additionally, I would like to thank my family members for their unwilling encouragement and help throughout the report.

Thank you.

Table of Contents

[1. Introduction 6](#_Toc162718234)

[1.1 The Problem 6](#_Toc162718235)

[1.2 The Solution 6](#_Toc162718236)

[2. Algorithm 7](#_Toc162718237)

[2.1 Pseudocode 7](#_Toc162718238)

[2.1.1 Precolation/\_\_init\_\_.py 7](#_Toc162718239)

[2.1.2 Precolation/helper\_functions.py 7](#_Toc162718240)

[2.1.3 Precolation/grid\_maker.py 11](#_Toc162718241)

[2.1.4 Precolation/txt.py 15](#_Toc162718242)

[2.1.5 Precolation/html.py 16](#_Toc162718243)

[2.1.6 main.py 17](#_Toc162718244)

[2.2 Classes 17](#_Toc162718245)

[2.2.1 Grid\_Maker() 17](#_Toc162718246)

[2.2.2 Random() 23](#_Toc162718247)

[2.2.3 Ok\_or\_not() 24](#_Toc162718248)

[2.2.4 HTML() 26](#_Toc162718249)

[2.2.4 Text() 28](#_Toc162718250)

[2.3 Functions 29](#_Toc162718251)

[2.3.1 grid\_condition() 29](#_Toc162718252)

[2.3.2 director\_creator() 30](#_Toc162718253)

[2.4 Packages Used 30](#_Toc162718254)

[2.4.1 sys 30](#_Toc162718255)

[2.4.2 random 30](#_Toc162718256)

[2.4.3 os 30](#_Toc162718257)

[2.4.4 datetime 30](#_Toc162718258)

[2.4.5 prettytable 31](#_Toc162718259)

[2.5 Project Structure 31](#_Toc162718260)

[2.5.1 Precolation/\_\_init\_\_.py 32](#_Toc162718261)

[2.5.2 Precolation/Grid\_Maker.py 32](#_Toc162718262)

[2.5.3 Precolation/helper\_function.py 32](#_Toc162718263)

[2.5.4 Precolation/html.py 32](#_Toc162718264)

[2.5.4 Precolation/txt.py 32](#_Toc162718265)

[2.5.4 main.py 32](#_Toc162718266)

[2.6 Explanation 32](#_Toc162718267)

[3 Assumptions 33](#_Toc162718268)

[4 Python Code 33](#_Toc162718269)

[4.1 Precolation/\_\_init\_\_.py 33](#_Toc162718270)

[4.2 Precolation/grid\_maker.py 34](#_Toc162718271)

[4.3 Precolation/helper\_functions.py 37](#_Toc162718272)

[4.4 Precolation/html.py 41](#_Toc162718273)

[4.5 Precolation/txt.py 42](#_Toc162718274)

[4.6 main.py 43](#_Toc162718275)

[5 Test Cases 43](#_Toc162718276)

[5.1 Valid Grid 43](#_Toc162718277)

[5.2 No Command Line Arguments (Default) 44](#_Toc162718278)

[5.3 Invalid Grid 44](#_Toc162718279)

[5.4 Impossible Percolation 45](#_Toc162718280)

[5.5 Minimum Grid Dimensions 46](#_Toc162718281)

[5.6 Maximum Grid Dimensions 46](#_Toc162718282)

[5.7 Text File Generation 47](#_Toc162718283)

[5.8 HTML File Generation 48](#_Toc162718284)

[6 Future Improvements 49](#_Toc162718285)

[7 Conclusion 49](#_Toc162718286)

Table of Figures

[Figure 1 18](#_Toc162719164)

[Figure 2 19](#_Toc162719165)

[Figure 3 20](#_Toc162719166)

[Figure 4 21](#_Toc162719167)

[Figure 5 22](#_Toc162719168)

[Figure 6 22](#_Toc162719169)

[Figure 7 23](#_Toc162719170)

[Figure 8 23](#_Toc162719171)

[Figure 9 23](#_Toc162719172)

[Figure 10 24](#_Toc162719173)

[Figure 11 24](#_Toc162719174)

[Figure 12 25](#_Toc162719175)

[Figure 13 25](#_Toc162719176)

[Figure 14 26](#_Toc162719177)

[Figure 15 26](#_Toc162719178)

[Figure 16 26](#_Toc162719179)

[Figure 17 27](#_Toc162719180)

[Figure 18 27](#_Toc162719181)

[Figure 19 28](#_Toc162719182)

[Figure 20 28](#_Toc162719183)

[Figure 21 29](#_Toc162719184)

[Figure 22 29](#_Toc162719185)

[Figure 23 29](#_Toc162719186)

[Figure 24 30](#_Toc162719187)

[Figure 25 32](#_Toc162719188)

[Figure 26 44](#_Toc162719189)

[Figure 27 44](#_Toc162719190)

[Figure 28 45](#_Toc162719191)

[Figure 29 46](#_Toc162719192)

[Figure 30 46](#_Toc162719193)

[Figure 31 47](#_Toc162719194)

[Figure 32 48](#_Toc162719195)

[Figure 33 49](#_Toc162719196)

# 1. Introduction

## 1.1 The Problem

The objective is to develop a Python program to simulate the process of percolation, which is the process of liquid slowly passing through a filter. There need to be two outputs of a text file (.txt) and an HTML file (.html) which contains a grid containing 2 random numbers between 10 and 99 and with random empty cells. Percolation is achievable in an entire column in the grid without empty cells, percolation isn’t achieved if the column has empty cells.

## 1.2 The Solution

The solution that is created is for the problem to simulate percolation where liquid flows through a filter, and the main objective of the simulation is to check if percolation is achievable or not. A grid containing random two digital numbers, with random empty cells. Percolation is achieved according to the condition that an entire column in the grid is without empty cells in turn, otherwise, percolation isn’t achieved. The program will output 2 files, a text file (.txt) as well as an HTML File (.html) allowing one to visualize the percolation process easily. The solution is a simulation system built using Python programming language.

# 2. Algorithm

The solution which is implemented by Python Programming language is stated below in the form of algorithm steps and with an explanation of how each aspect of the program modules, classes, and functions.

## 2.1 Pseudocode

### 2.1.1 Precolation/\_\_init\_\_.py

1. IMPORT sys
2. IMPORT random
3. IMPORT os 4. IMPORT datetime

5.

1. TRY:
2. FROM prettytable IMPORT PrettyTable
3. EXCEPT:
4. os.system("pip install prettytable") 10. FROM prettytable IMPORT PrettyTable
5. ELSE:
6. os.system("python3 -m pip install prettytable")
7. FROM prettytable IMPORT PrettyTable
8. finally:
9. os.system("pip3 install prettytable") 16. FROM prettytable IMPORT PrettyTable

17.

18.

1. FROM Percolation.helper\_functions IMPORT \*
2. FROM Percolation.grid\_maker IMPORT \*
3. FROM Percolation.txt IMPORT \*
4. FROM Percolation.html IMPORT \*

### 2.1.2 Precolation/helper\_functions.py

1. FROM Percolation IMPORT random, PrettyTable, os

2.

3.

1. DEFINE CLASS Random:
2. @staticmethod
3. DEFINE FUNCTION generate\_random\_number(start: int, end: int) -> int:
4. """
5. Generate a random integer number between start and end (inclusive).

9.

1. Args:
2. start (int): The minimum value of the random integer.
3. end (int): The maximum value of the random integer (inclusive).

13.

1. Returns:
2. int: A random integer number between start and end.
3. """
4. RETURN random.randint(
5. start, end
6. ) # Create a random integer number between start and end

20.

1. @staticmethod
2. DEFINE FUNCTION select\_choice(choices: list, probability: tuple[float]) -> list:
3. """
4. Select a random choice FROM a list based on a given probability distribution.

25.

1. Args:
2. choices (list): A list of possible choices.
3. probability (tuple[float]): A tuple of probabilities corresponding to each choice IN `choices`. The sum of all probabilities must be equal to 1.

29.

1. Returns:
2. list: A single randomly selected choice FROM `choices` based on the given probability distribution.

32.

1. Raises:
2. ValueError: If the length of `choices` and `probability` do not match.
3. """
4. IF len(choices) EQUALS len(probability):
5. RETURN random.choices(
6. choices, weights=probability
7. ) # Return a random choice FROM the given probability distribution
8. raise ValueError(
9. f"The list `choices` and tuple `probability` must have the same length. {choices}

({len(choices)}) != {probability} ({len(probability)})"

1. ) # Create a error IF the length of choices and length of probability is not equal

43.

44.

1. DEFINE CLASS Ok\_or\_not:
2. """
3. This DEFINE CLASS takes a 2D list as INPUT and determines whether each column of the list contains all unique elements. 48. """

49.

1. DEFINE FUNCTION \_\_init\_\_(self, grid: list) -> None:
2. """
3. Args:
4. grid (list): A 2D list of integers. Each row represents a column of the original list.
5. """
6. SET self.grid TO grid
7. SET self.t TO PrettyTable(header=False)
8. SET self.col\_length TO len(self.grid[0])
9. SET self.cols\_data TO {
10. i: [] FOR i IN range(self.col\_length)
11. } # create a dictionary WITH column index id and WITH a value as a list
12. SET self.filter TO lambda cols\_data: [
13. "NO" IF "" IN cols\_data[col] else "OK" FOR col IN cols\_data 63. ] # filter the columns to see IF there are empty cell IN a column

64.

1. DEFINE FUNCTION generate(self) -> list:
2. """
3. This function generates a table showing whether each column of the INPUT list contains all unique elements.
4. The output is a string containing the table IN HTML format.

69.

1. Returns:
2. str: The table showing whether each column of the INPUT list contains all unique elements.
3. """
4. FOR row IN self.grid:
5. SET row TO row.copy()
6. FOR ele IN row:
7. SET idx TO row.index(ele)
8. self.cols\_data[idx].append(
9. ele
10. ) # Add each element to each column
11. SET row[idx] TO None
12. SET list\_ok\_or\_not TO self.filter(self.cols\_data) # Filter all the columns
13. self.t.add\_row(list\_ok\_or\_not) # Add to the self.t table
14. RETURN list\_ok\_or\_not

84.

1. DEFINE FUNCTION get\_html(self) -> str:
2. """
3. This function RETURNs the table showing whether each column of the INPUT list contains all unique elements 88. IN HTML format.

89.

1. Returns:
2. str: The table showing whether each column of the INPUT list contains all unique elements IN HTML format.
3. """
4. RETURN self.t.get\_html\_string(header=False) # get a html output of the table

94.

1. DEFINE FUNCTION get\_string(self) -> str:
2. """
3. This function RETURNs the table showing whether each column of the INPUT list contains all unique elements 98. IN string format.

99.

1. Returns:
2. str: The table showing whether each column of the INPUT list contains all unique elements IN string format.
3. """
4. RETURN self.t.get\_string(header=False) # get a string output of the table

104.

105.

1. DEFINE FUNCTION grid\_condition(dims: str) -> tuple:
2. """
3. This function takes a string INPUT IN the form of "x" separated row and column dimensions and RETURNs a tuple of integers representing the dimensions.
4. The function ensures that the INPUT is IN the correct format and that the row and column dimensions are within the specified range.
5. If the INPUT is not IN the correct format or the dimensions are outside the specified range, the function RETURNs a default value of (5, 5).

111.

1. Args:
2. dims (str): A string IN the form of "x" separated row and column dimensions.

114.

1. Returns:
2. tuple: A tuple of integers representing the row and column dimensions.
3. """
4. SET split TO dims.split(dims[int(len(dims)/2)]) # Splitting the dim by 'middle' character
5. IF any([s.isnumeric() FOR s IN split]): # checking IF the picked character is a number
6. SET split TO dims.split('x') # then try and split FROM 'x'
7. IF len(split) EQUALS 2: # Checking conditions
8. SET rows, cols TO split
9. IF rows.isnumeric() and cols.isnumeric(): 124. SET rows, cols TO list(map(int, split)) 125. IF 3 <= rows <= 9 and 3 <= cols <= 9:
10. RETURN rows, cols
11. OUTPUT(‘The entered grid size is invalid… The default 5x5 Grid is used’)
12. RETURN 5, 5 # Returning the default grid sizes

129.

130.

1. DEFINE FUNCTION director\_creator(directory: str) -> bool:
2. """
3. Creates a directory WITH the given name IN the current working directory.

134.

* 1. Parameters:
  2. directory (str): The name of the directory to create.

137.

* 1. Returns:
  2. bool: True IF the directory was created, False IF the directory already exists.
  3. """
  4. IF not os.path.isdir(f"./{directory}"): # checking IF the directory doesnt exist
  5. os.mkdir(f"./{directory}") # making the directory
  6. RETURN True
  7. RETURN False

### 2.1.3 Precolation/grid\_maker.py

1. FROM Percolation IMPORT Random, random, PrettyTable

2.

3.

1. DEFINE CLASS Grid\_Maker:
2. SET def \_\_init\_\_(self, rows: int, cols: int, perc\_of\_empty: float TO 0.1) -> None:
3. """
4. Initialize a new instance of the Grid\_Maker class.

8.

1. Args:
2. rows (int): The number of rows IN the grid.
3. cols (int): The number of columns IN the grid.
4. perc\_of\_empty (float, optional): The percentage of cells that should be left empty. Defaults to 0.1.
5. """
6. SET self.grid TO []
7. SET self.rows TO rows
8. SET self.cols TO cols
9. SET self.tot TO rows \* cols
10. SET self.r TO Random() # Creating a instance of Random() object
11. SET self.perc\_of\_empty TO perc\_of\_empty
12. SET self.coordinate\_cols TO lambda x: [
13. i[1] FOR i IN x
14. ] # getting the first index of `x` list

23.

1. DEFINE FUNCTION \_\_empty\_cell(self, row: int, col: int) -> bool:
2. """
3. Checks IF the given row and column indices are within the bounds of the grid,
4. and IF the cell at that location is not already empty. If both conditions are true, 28. the method sets the cell value to an empty string and RETURNs True, indicating that

29. the cell was successfully marked as empty. Otherwise, it RETURNs False.

30.

1. Args:
2. row (int): The row index of the cell to be checked.
3. col (int): The column index of the cell to be checked.

34.

1. Returns:
2. bool: True IF the cell was successfully marked as empty, False otherwise.
3. """
4. IF (
5. row
6. <= self.rows # Checking IF the row given is less than or equal to self.rows 41. or col
7. <= self.cols # Checking IF the col given is less than or equal to self.cols
8. and ""
9. not IN [
10. row[col] FOR row IN self.grid
11. ] # Make sure that column doesn't have empty cells 47. and self.grid != []
12. ):
13. SET self.grid[row - 1][col - 1] TO ""
14. RETURN True
15. RETURN False

52.

1. DEFINE FUNCTION \_\_check\_if\_all\_columns\_have\_empty\_cell(self) -> bool:
2. """
3. This function checks IF any of the columns have empty space left to remove

56.

1. Args:
2. None

59.

1. Returns:
2. bool: A boolean checking the `cols\_check` list and IF any of them has a True, True will be RETURNed other wise False would be RETURNed
3. """
4. SET cols\_check TO [] 64. FOR col IN range(self.cols):
5. SET empty TO "" IN [
6. row[col] FOR row IN self.grid
7. ]
8. cols\_check.append(False IF empty else True) 69. RETURN any(cols\_check)

70.

1. DEFINE FUNCTION generate\_cells\_to\_empty(self) -> list:
2. """
3. This function generates a list of random row and column indices that represent
4. the locations of the cells that should be left empty IN the grid. It does so by
5. randomly selecting a row and column index, and checking IF the cell at that location
6. is already empty or not. If the cell is empty, the function marks it as empty by
7. setting its value to an empty string, and decrements the count of the remaining
8. empty cells. The function continues to select random indices UNTIL the desired
9. number of empty cells has been reached.

80.

1. Args:
2. None

83.

1. Returns:
2. list: A list of tuples, where each tuple represents the row and column indices
3. of a cell that should be left empty.
4. """
5. SET empty\_no\_of\_cell TO int(self.tot \* self.perc\_of\_empty)
6. SET empty\_no\_of\_cell TO (
7. empty\_no\_of\_cell - self.cols
8. IF empty\_no\_of\_cell > self.cols
9. else self.cols - empty\_no\_of\_cell
10. ) # Calculate the number of cells that has to be emptied
11. SET coordinates TO []
12. SET empty\_cells\_in\_cols TO self.\_\_check\_if\_all\_columns\_have\_empty\_cell()
13. WHILE empty\_no\_of\_cell != 0 and empty\_cells\_in\_cols:
14. SET row, col TO random.randint(1, self.rows), random.randint(
15. 1, self.cols
16. ) # Select a random row and column
17. IF col not IN self.coordinate\_cols(coordinates):
18. SET deleted\_status TO self.\_\_empty\_cell(
19. row, col
20. ) # getting the status of emptying a cell 104. IF deleted\_status:
21. empty\_no\_of\_cell -= 1
22. coordinates.append((row, col)) # add the coordinates
23. SET empty\_cells\_in\_cols TO self.\_\_check\_if\_all\_columns\_have\_empty\_cell()
24. RETURN coordinates

109.

1. DEFINE FUNCTION make\_grid(self) -> list:
2. """
3. This function creates a grid of random numbers IN the range of 10 to 99.
4. It iterates over the number of rows and columns IN the grid and appends a random number to each cell.
5. The function RETURNs the grid as a list of lists.
6. """
7. FOR \_ IN range(self.rows):
8. SET row TO []
9. FOR \_ IN range(self.cols):
10. row.append(
11. self.r.generate\_random\_number(10, 99)
12. ) # add the random number generated to the list
13. self.grid.append(row) # add the random number list generated to the list
14. RETURN self.grid

124.

* 1. DEFINE FUNCTION create\_table(self) -> PrettyTable:
  2. """
  3. This function creates a table using PrettyTable

128.

129. Args: 130. None

131.

* 1. Returns:
  2. PrettyTable: a table WITH all of the data IN the 2D grid `self.grid`
  3. """
  4. SET table TO PrettyTable(header=False) # Create a table 136. FOR row IN self.grid:
  5. table.add\_row(row) # Add a row
  6. RETURN table 139.
  7. DEFINE FUNCTION generate\_string(
  8. self, 142. ) -> str:
  9. """
  10. This function generates a string representation of the grid.

145.

* 1. Returns:
  2. str: A string representation of the grid.
  3. """
  4. RETURN str(self.create\_table().get\_string()) # Return a string without the header

150.

* 1. DEFINE FUNCTION generate\_html(self):
  2. """
  3. This function generates an HTML representation of the grid

154.

* 1. Returns:
  2. str: An HTML representation of the grid.
  3. """
  4. RETURN str(self.create\_table().get\_html\_string()) # Return the string without the header

159.

* 1. DEFINE FUNCTION grid\_maker(self) -> list:
  2. self.make\_grid() # Make a grid
  3. self.generate\_cells\_to\_empty() # Empty random grid
  4. RETURN self.grid # send the final grid

### 2.1.4 Precolation/txt.py

1. FROM Percolation IMPORT Grid\_Maker, Ok\_or\_not, director\_creator

2.

3.

4. DEFINE CLASS Text:

5.

1. DEFINE FUNCTION \_\_init\_\_(self, grid: Grid\_Maker, ok\_or\_not: Ok\_or\_not, file\_name: str) -> None:
2. """
3. Initialize a new instance of the Text class.

9.

1. Args:
2. grid (Grid\_Maker): an instance of the Grid\_Maker DEFINE CLASS that generates the text grid
3. ok\_or\_not (Ok\_or\_not): an instance of the Ok\_or\_not DEFINE CLASS that generates the "ok" or "not ok" message
4. file\_name (str): the name of the file to be created
5. """
6. SET self.grid TO grid
7. SET self.ok\_or\_not TO ok\_or\_not
8. SET self.file\_name TO f"./{file\_name}/{file\_name}.txt" 18. director\_creator(file\_name)

19.

1. DEFINE FUNCTION create\_full\_grid(self) -> str:
2. """
3. This function combines the output of the grid and the ok\_or\_not functions to create a full text grid.

23.

1. Returns:
2. str: the combined output of the grid and the ok\_or\_not functions
3. """
4. RETURN (
5. self.grid.generate\_string() + "\n" + self.ok\_or\_not.get\_string() 29. ) # create the string

30.

1. DEFINE FUNCTION create\_file(self) -> str:
2. """
3. This function creates a text file WITH the combined output of the grid and the ok\_or\_not functions.
4. Returns:
5. str: the full text grid WITH the "ok" or "not ok" message

36.

1. """
2. SET txt TO self.create\_full\_grid() # getting the full string of the grid
3. WITH open(self.file\_name, "w") as f:
4. f.write(txt) # add to the file
5. RETURN txt

### 2.1.5 Precolation/html.py

1. FROM Percolation IMPORT Grid\_Maker, Ok\_or\_not, director\_creator

2.

3.

1. DEFINE CLASS HTML:
2. DEFINE FUNCTION \_\_init\_\_(self, grid: Grid\_Maker, ok\_or\_not: Ok\_or\_not, file\_name: str) -> None:
3. """
4. Initialize the HTML class.

8.

1. Args:
2. grid (Grid\_Maker): The grid maker object.
3. ok\_or\_not (Ok\_or\_not): The ok or not object.
4. file\_name (str): The file name.
5. """
6. SET self.grid TO grid
7. SET self.ok\_or\_not TO ok\_or\_not
8. SET self.file\_name TO f"./{file\_name}/{file\_name}.html" # the directory
9. # calling the director\_creator() helper function 18. director\_creator(file\_name)

19.

1. @staticmethod
2. DEFINE FUNCTION create\_html\_code(elements: str, name: str="Grid") -> str:
3. RETURN f"""
4. <!DOCTYPE html>
5. <html lang="en">
6. <head>
7. <title>{name}</title>
8. </head>
9. <body>{elements}</body>
10. </html>
11. """ # RETURN the default html code template

31.

1. DEFINE FUNCTION create\_full\_grid(self) -> str:
2. """
3. This function generates the full HTML code FOR the Percolation simulation.

35.

1. Returns:
2. str: The full HTML code FOR the Percolation simulation.
3. """
4. RETURN (
5. self.grid.generate\_html() + self.ok\_or\_not.get\_html()
6. ) # RETURNing the combination of grid and ok or not

42.

1. DEFINE FUNCTION create\_file(self) -> str:
2. """
3. This function generates the full HTML code FOR the Percolation simulation.

46.

1. Returns:
2. str: The full HTML code FOR the Percolation simulation.
3. """
4. SET code TO self.create\_html\_code(
5. self.create\_full\_grid()
6. ) # generating the html code 53. WITH open(self.file\_name, "w") as f:
7. f.write(code) # inserting the code into the file
8. RETURN code

### 2.1.6 main.py

1. FROM Percolation IMPORT grid\_condition, Grid\_Maker, Ok\_or\_not, HTML, Text, sys, datetime

2.

1. IF \_\_name\_\_ EQUALS "\_\_main\_\_":
2. SET rows, cols TO grid\_condition(sys.argv[-1])
3. SET gm TO Grid\_Maker(rows, cols)
4. SET two\_dim\_grid TO gm.grid\_maker()
5. SET oon TO Ok\_or\_not(two\_dim\_grid)
6. SET ok\_or\_not\_list TO oon.generate()
7. SET file\_name TO datetime.datetime.now().strftime("%Y\_%m\_%d\_%H%M")
8. HTML(gm, oon, file\_name).create\_file()
9. SET t TO Text(gm, oon, file\_name).create\_file()
10. OUTPUT(t)

## 2.2 Classes

### 2.2.1 Grid\_Maker()

The class `Grid\_Maker()` is designed to generate and manipulate grids for the use of simulating the percolation process. The class contains the following functions:

#### 2.2.1.1 \_\_init\_\_()

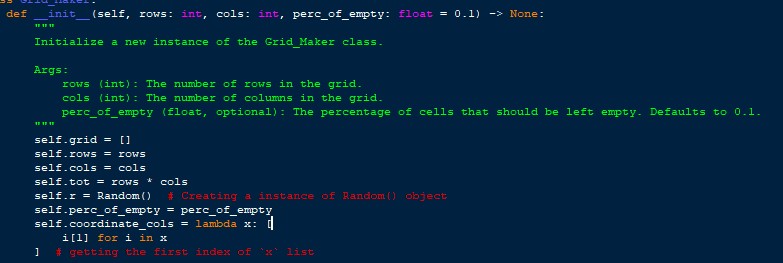


Figure 1

This function is the initialization function which runs when an instance of the class is created.

The function takes in the parameters of `rows` which is the no. of rows expected from the grid, `cols` which is the columns expected from the grid, and `perc\_of\_empty` which is the percentage of cells that should be empty, and it is default at 0.1 (which means 10%). The function returns Nothing. The data gathered is added to the object attributes and self.grid which is an empty list that will contain the 2D array later on, self.r an instance of the Random() class,self.tot which finds the total number of elements in the grid, self.coordinate\_cols which is a lambda function which takes the first element of the list `x` passed into the function those attributes are also added.

#### 2.2.1.2 \_\_empty\_cell ()

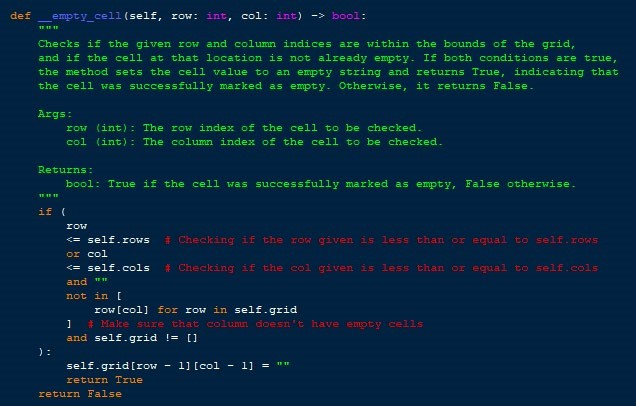


Figure 2

Which is a private function (hidden function) that is used to empty a cell after the grid is generated. The function takes in 2 parameters `row` and `col` which both are of the data type `int` and the function returns a Boolean which if `True` means that the cell has been emptied, and `False` means that the cell hasn’t been emptied due to the conditions not matching up. The condition for emptying a cell is: The `self.rows` has to be bigger or equal to the `row` variable, `self.cols` has to be bigger or equal to `col`, “” can't be in the column previously, this is to check the assumption that a column can only have 1 empty cell and self.grid can't be an empty list this is to make sure that the make\_grid() function has been already executed and the self.grid function isn’t empty. If the conditions are accepted the cell is emptied or “” is set as the value.

#### 2.2.1.3 \_\_check\_if\_all\_columns\_have\_empty\_cell()

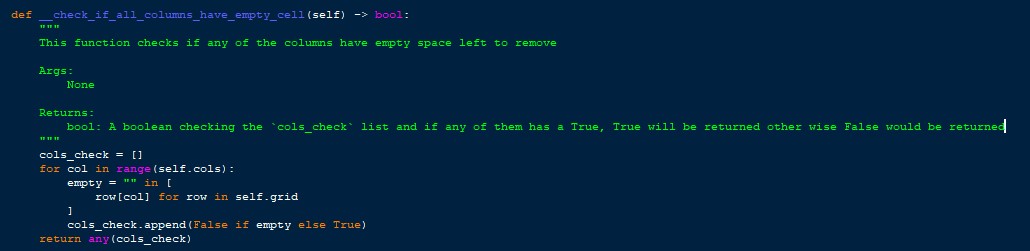


Figure 3

The function checks if there is a space in any of the columns to remove. First, a list is initiated which is used to add Boolean data type and at the end, we check if any of the added bools are True if they are True is returned otherwise False is returned. The function iterates through all of the `self.cols` and checks whether there is space if it is False is added to the list otherwise True is added to the list and at the end the list is checked and the condition is returned.

#### 2.2.1.4 generate\_cells\_to\_empty()

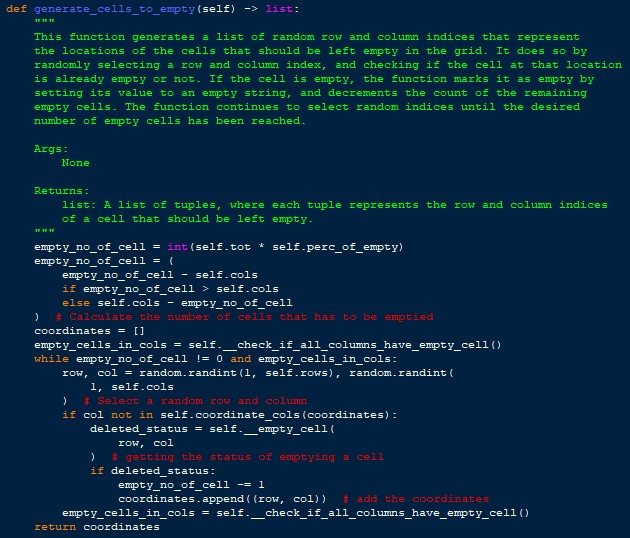


Figure 4

This function has no parameters, and it returns a list that contains all the coordinates that have been emptied, which aren’t being used but are added for future improvements of the project. The function first generates the number of empty cells that should be in the grid using the

`self.prec\_of\_empty`and `self.tot`, then it is re-calculated by checking if the empty\_no\_of\_cells is higher than the number of cols then the empty\_no\_of\_cells is subtracted by the `self. cols` variable otherwise `self.cols` is subtracted by empty\_no\_of\_cells. Then the coordinates list is created and we use the ` \_\_check\_if\_all\_columns\_have\_empty\_cell()` function to make sure that there are columns that have no empty spaces and if both conditions of `empty\_no\_of\_cell` aren’t 0 and there are columns without empty cells a while loop is started which first creates row and col which are random numbers and then we check if an element from the col hasn’t been removed beforehand using the self.coordinates\_cols lambda function and if the condition is true then the row and column are passed through to the self.\_\_empty\_cell() function and according to the response if the cell was deleted successfully then the empty\_no\_of\_cell is subtracted and then the coordinates (in a tuple) are added to the coordinates list. Then the `empty\_cells\_in\_cols` variable is refreshed and the next iteration will start. Finally, the coordinates of the removed cells will be returned.

#### 2.2.1.5 make\_grid()

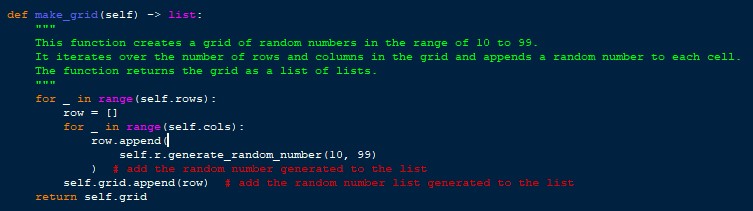


Figure 5

This function creates the elements of the 2D grid using 2 for loops and using the `self.r` Random class initiation and by using its `generate\_random\_number()` function. Then each row is added to the self. grid list and the grid is returned as well.

#### 2.2.1.6 create\_table()

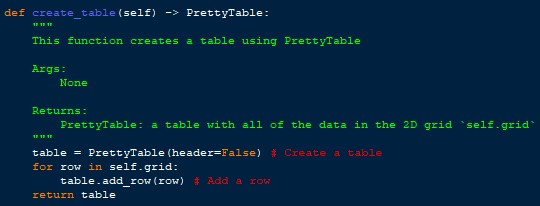


Figure 6

The function first initiates a PrettyTable instance and then adds all of the rows in the self. grid 2D list. Then finally it returns to the table created.

#### 2.2.1.7 generate\_string()

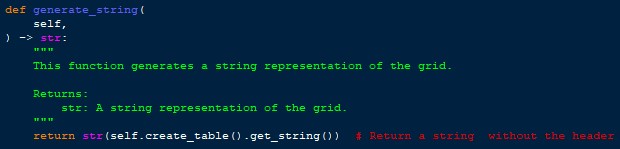


Figure 7

Create a table using the `self.create\_table()` function and then use the `.get\_string()` function on the returned PrettyTable instance. The function returns a string representation of the grid.

#### 2.2.1.8 generate\_html()

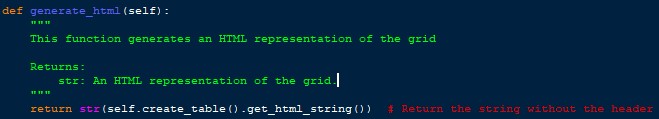


Figure 8

Create a table using the `self.create\_table()` function and then use the `.get\_html()` function on the returned PrettyTable instance. The function returns an HTML representation of the grid.

#### 2.2.1.9 grid\_maker()

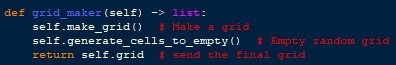


Figure 9

The function just calls the self.make\_grid() function to create a grid of random numbers and then calls the self.generate\_cells\_to\_empty() to make random numbers empty and then finally returns the self.grid 2D grid.

### 2.2.2 Random()

This function provides static methods to generate random numbers and select choices using a probability distribution. The function has static methods due to it being easier to manage and it is bundled together if further developments are made the Random() class can develop with it.

#### 2.2.2.1 generate\_random\_number()

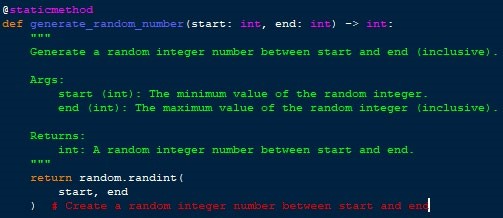


Figure 10

The function takes in 2 parameters `start` and `end` which are the lowest possible and highest possible numbers that can be randomly picked. Then the `random` library’s `random. randint()` function generates a random number between the start and end, and finally, the random number is returned.

#### 2.2.2.2 select\_choice()

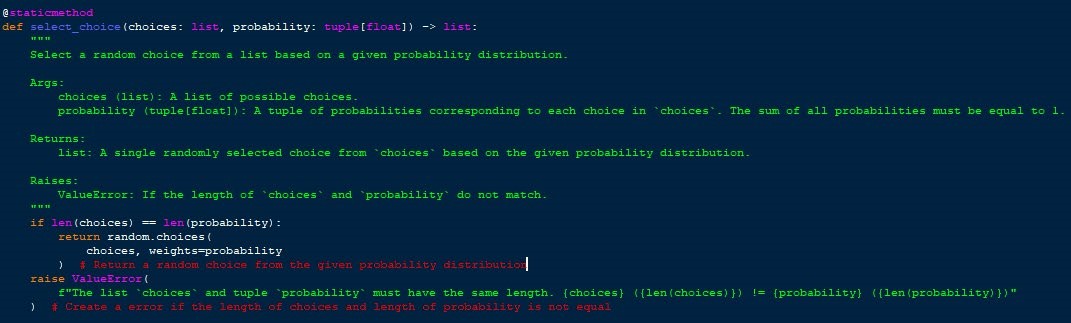


Figure 11

The function takes in a choice (list) and weights (tuple with float values) and we first check if their lengths are the same and if not then a ValueError is raised. If the lengths match up then the choices and weights are passed into the `random.choices()` function of the `random` library in Python.

### 2.2.3 Ok\_or\_not()

Takes in a 2D grid and produces an HTML and string which tells whether or not each column of the grid has achieved percolation or not.

#### 2.2.3.1 \_\_init\_\_()

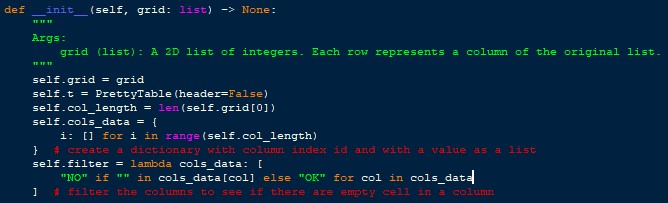


Figure 12

This function is initiated with the class itself, the function takes in the grid (2D) and that is set as a global attribute, and then an instance of the PrettyTable is added as well, and the length of the column is saved as well, create a dictionary with an index and a list for all of the cols, and a lambda function is established which checks a specific column and checks if there is an “” in the column if there is “NO” is added to the list if not “OK” is added to the list.

#### 2.2.3.2 generate()

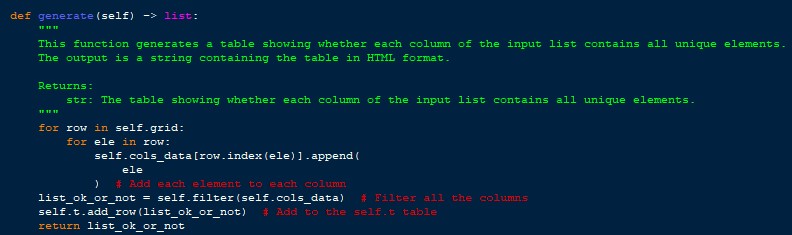


Figure 13

The function doesn’t have any parameters, it returns a list of whether the columns have achieved percolation or not. The function first goes through the entire grid using 2 loops and adds all of the elements into the self.cols\_data dictionary and then use the self.filter() function on the self.cols\_data and then add the list returned by the function to the pretty table instance.

#### 2.2.3.3 get\_html()

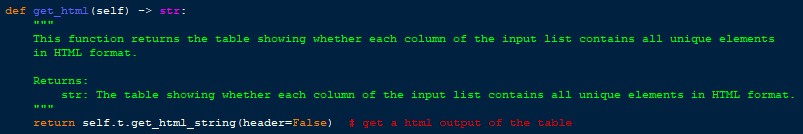


Figure 14

`get\_html()` function produces a html string using the `self.t` PrettyTable. It returns the string with the HTML table.

#### 2.2.3.4 get\_string()

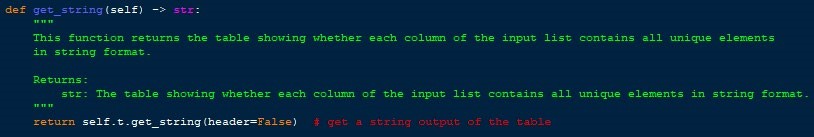


Figure 15

`get\_string()` function produces a string using the `self.t` PrettyTable and returns the string.

### 2.2.4 HTML()

This class is responsible for creating an HTML file of the percolation simulation, and at the end, the class produces an HTML file at a specified file location with the current datetime.

#### 2.2.3.1 \_\_init\_\_()

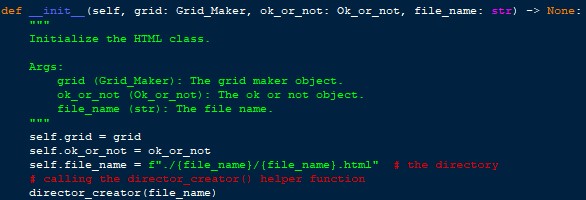


Figure 16

This function initiates with the creation of an instance of the class as well. This function takes in the grid in the form of the Grid\_Maker class instance ok\_or\_not in the form of the Ok\_or\_not class instance, and the file\_name as well. The `director\_creator()` function is called to make sure that the directory with the file name already exists. Nothing is returned in the function.

#### 2.2.3.2 create\_html\_code()

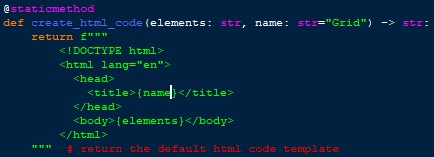


Figure 17

The `create\_html\_code()` contains a pre-defined string that has the basic structure of an HTML file, and it fills the structure with the parameters of `elements` and `name` which contain the body and title respectively. Finally, the function returns the structure with the updated body and title.

#### 2.2.3.3 create\_full\_grid()

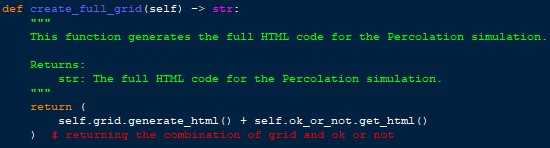


Figure 18

This function generates the entire HTML code by using the `Grid\_Maker` and `Ok\_or\_not` instances.

#### 2.2.3.4 create\_file()

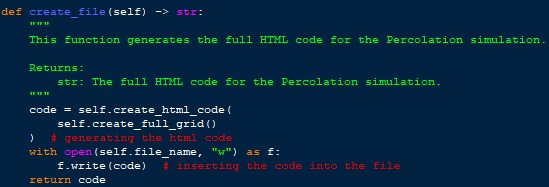


Figure 19

This function utilizes both other functions (`create\_html\_code` and `create\_full\_grid`) and creates the final code. The function first creates the full grid (using the `create\_full\_grid`) and then passes the return from the function into `create\_html\_code` and then finally writes the created html code into the file (self.file\_name) and returns the code as well.

### 2.2.4 Text()

The `Text` class combines the grid and ok or not tables into the text format and inserts it into a file.

#### 2.2.4.1 \_\_init\_\_()

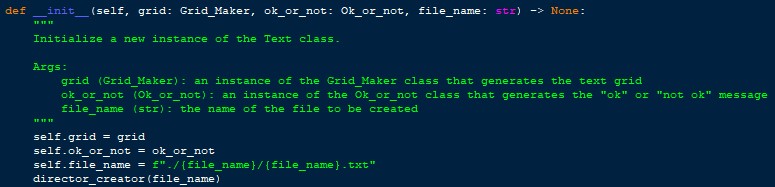


Figure 20

This function runs when the Text class is initiated and it takes in 3 parameters which are: grid which is an instance of the Grid\_Maker class, ok\_or\_not which is an instance of the Ok\_or\_not, and file\_name which is the file name that the string should be saved in. This function runs the `director\_creator()` function to make sure that a directory with the file name is created.

#### 2.2.4.2 create\_full\_grid()

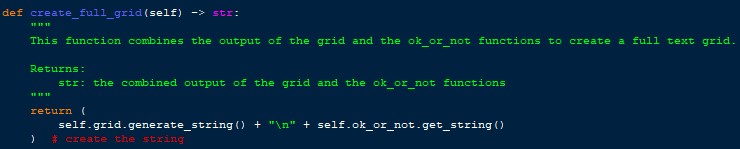


Figure 21

`create\_full\_grid()` function gets the string from both the Grid\_Maker and Ok\_or\_not class instnacnes and combines them and returns them.

#### 2.2.4.3 create\_file()

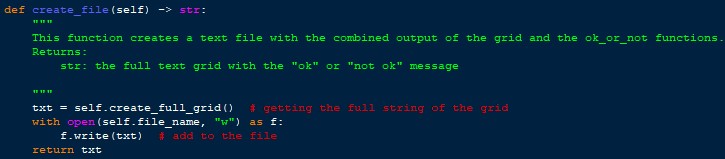


Figure 22

This function gets the combined string from `self.create\_full\_grid()` and then writes the returned string into the text file.

## 2.3 Functions

### 2.3.1 grid\_condition()

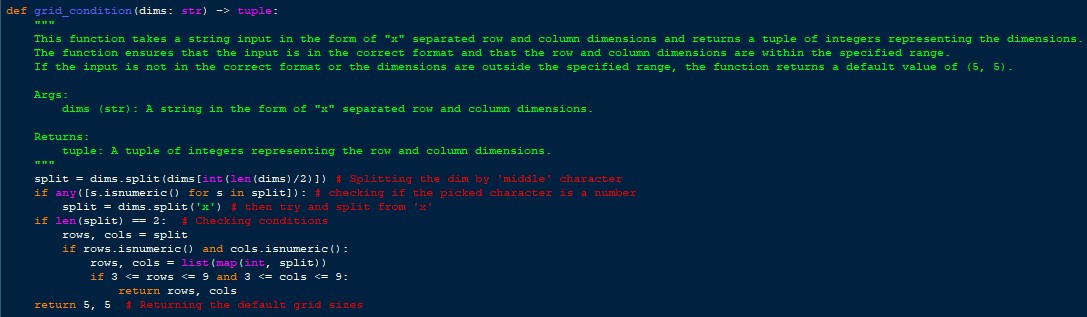


Figure 23

This function is used to make sure that the entered grid size fits the criteria. We try and split the middle event so that in the case of the user entering a space instead of an `x` the program will still work, and in the worst-case scenario that it isn’t the entered console argument is split by `x`, and then we check for the length of the split list and then check if they match the conditions of both the rows and columns being bigger than or equal to 3 and less than or equal to 9. The parameter is dims which is the console argument from the user and a tuple is returned.

### 2.3.2 director\_creator()

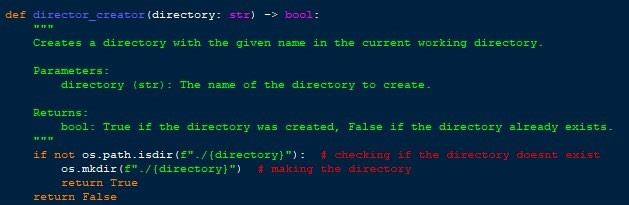


Figure 24

The function `director\_creator()` checks whether a directory already exists if not then it creates one and returns True, if it already exists it returns False. The parameter is the directory, and the return is a Boolean.

## 2.4 Packages Used

### 2.4.1 sys

The `sys` package is used in the Python program to get the console arguments from the user, and then it passes through to the `grid\_condition()` function and finally, we get the rows and cols that the user entered.

### 2.4.2 random

The `random` package is used in many occasions throughout the program. It is used in the following situations: Finding a random coordinate in the grid to empty,

Random().generate\_random\_number() to create a random number and random.randint() is used, Random().select\_choice() to select a choice using probability where `random.choices()` is used.

### 2.4.3 os

`os` package is used in the program to make sure that the `prettytable` library is installed using `os.system()` and in the `director\_creator()` helper\_function where `os.path.isdir()` and `os.mkdir()` is used.

### 2.4.4 datetime

The `datetime` package is used to create the file name of the current time for the .html and .txt file.

### 2.4.5 prettytable

The `prettytable` package is one of the most important and useful packages used in the program, the prettytable is used to create a table with a structure it is extremely easy to use and flexible. The `prettytable` is used in Grid\_Maker().create\_table(), Ok\_or\_Not() initialization. All of the tables throughout the program use prettytable and its .get\_string() and .get\_html() are used for the filesaving functions.

## 2.5 Project Structure

The following is the project structure that has been used for this program.

Percolation/

│

│── Percolation/

│ |-── \_\_init\_\_.py

│ |-── Grid\_Maker.py

│ |-──helper\_functions.py

│ |-──html.py

│ |-──txt.py

│

└── main.py

### 2.5.1 Precolation/\_\_init\_\_.py

This file marks the directory as a Python module. This is like \_\_init\_\_ in a Class and the \_\_init\_\_.py file runs when we use Percolation.

### 2.5.2 Precolation/Grid\_Maker.py

This file contains the Grid\_Maker class used to generate grids.

### 2.5.3 Precolation/helper\_function.py

`helper\_function.py` contains multiple classes of `Random` which has all of the aspects of randomization, `Ok\_or\_not` class used to generate a table regarding whether the percolation of the columns in the grid is achieved or not, `grid\_condition()` which makes sure that entered grid dimensions are acceptable and `director\_creator()` used to create a directory if it doesn’t already exist.

### 2.5.4 Precolation/html.py

This file contains the class `HTML` which is used to save the .html file of the grid and ok or not.

### 2.5.4 Precolation/txt.py

This file contains the class `Text` which is used to save the .txt file of the grid and ok or not.

### 2.5.4 main.py

This is the main file that combines all the modules and creates the grid.

## 2.6 Explanation

A screenshot of a computer program

Description automatically generated

Figure 25

When the main.py file runs the first takes the `sys.argv[-1]` and passes it through to the `grid\_condition()` and the function will return the rows and cols, then the rows and cols are passed through to the `Grid\_Maker()` and we execute the `Grid\_Maker().grid\_maker()` function which produces a two dimensional grid, then we pass the two dimentainal grid through to `Ok\_or\_not()`where we call the `Ok\_or\_not().generate()` function to check if the columns in the grid (2D) has achieved percolation or not and it returns a list if percolation succeed or not for each column. Then we get the current datetime and then pass along the `Grid\_Maker()` instance and `Ok\_or\_not()` instance and the `file\_name` which is the currentdatetime to `HTML()` and call `.create\_file()` which creates the .html file then we do the same process to the `Text()` class and call `.create\_file()` then the Text File will be created. Finally, the table is printed as the output.

# 3 Assumptions

The listed below are assumptions that were made about the solution when making the program:

1. Each Column Can Have Only 1 Empty Cell

It is assumed that every column in the grid can have at the most 1 empty cell, this assumption was made due to all the Grids in the Course Work Specification Containing only 1 Cell Maximum Per Column.

# 4 Python Code

## 4.1 Precolation/\_\_init\_\_.py

import sys

import random

import os

import datetime

try:

from prettytable import PrettyTable

except:

os.system("pip install prettytable")

from prettytable import PrettyTable

else:

os.system("python3 -m pip install prettytable")

from prettytable import PrettyTable

finally:

os.system("pip3 install prettytable")

from prettytable import PrettyTable

from Percolation.helper\_functions import \*

from Percolation.grid\_maker import \*

from Percolation.txt import \*

from Percolation.html import \*

## 4.2 Precolation/grid\_maker.py

from Percolation import Random, random, PrettyTable

class Grid\_Maker:

def \_\_init\_\_(self, rows: int, cols: int, perc\_of\_empty: float = 0.1) -> None:

"""

Initialize a new instance of the Grid\_Maker class.

Args:

rows (int): The number of rows in the grid.

cols (int): The number of columns in the grid.

perc\_of\_empty (float, optional): The percentage of cells that should be left empty. Defaults to 0.1.

"""

self.grid = []

self.rows = rows

self.cols = cols

self.tot = rows \* cols

self.r = Random() # Creating a instance of Random() object

self.perc\_of\_empty = perc\_of\_empty

self.coordinate\_cols = lambda x: [

i[1] for i in x

] # getting the first index of `x` list

def \_\_empty\_cell(self, row: int, col: int) -> bool:

"""

Checks if the given row and column indices are within the bounds of the grid,

and if the cell at that location is not already empty. If both conditions are true,

the method sets the cell value to an empty string and returns True, indicating that

the cell was successfully marked as empty. Otherwise, it returns False.

Args:

row (int): The row index of the cell to be checked.

col (int): The column index of the cell to be checked.

Returns:

bool: True if the cell was successfully marked as empty, False otherwise.

"""

if (

row

<= self.rows # Checking if the row given is less than or equal to self.rows

or col

<= self.cols # Checking if the col given is less than or equal to self.cols

and ""

not in [

row[col] for row in self.grid

] # Make sure that column doesn't have empty cells

and self.grid != []

):

self.grid[row - 1][col - 1] = ""

return True

return False

def \_\_check\_if\_all\_columns\_have\_empty\_cell(self) -> bool:

"""

This function checks if any of the columns have empty space left to remove

Args:

None

Returns:

bool: A boolean checking the `cols\_check` list and if any of them has a True, True will be returned other wise False would be returned

"""

cols\_check = []

for col in range(self.cols):

empty = "" in [

row[col] for row in self.grid

]

cols\_check.append(False if empty else True)

return any(cols\_check)

def generate\_cells\_to\_empty(self) -> list:

"""

This function generates a list of random row and column indices that represent

the locations of the cells that should be left empty in the grid. It does so by

randomly selecting a row and column index, and checking if the cell at that location

is already empty or not. If the cell is empty, the function marks it as empty by

setting its value to an empty string, and decrements the count of the remaining

empty cells. The function continues to select random indices until the desired

number of empty cells has been reached.

Args:

None

Returns:

list: A list of tuples, where each tuple represents the row and column indices

of a cell that should be left empty.

"""

empty\_no\_of\_cell = int(self.tot \* self.perc\_of\_empty)

empty\_no\_of\_cell = (

empty\_no\_of\_cell - self.cols

if empty\_no\_of\_cell > self.cols

else self.cols - empty\_no\_of\_cell

) # Calculate the number of cells that has to be emptied

coordinates = []

empty\_cells\_in\_cols = self.\_\_check\_if\_all\_columns\_have\_empty\_cell()

while empty\_no\_of\_cell != 0 and empty\_cells\_in\_cols:

row, col = random.randint(1, self.rows), random.randint(

1, self.cols

) # Select a random row and column

if col not in self.coordinate\_cols(coordinates):

deleted\_status = self.\_\_empty\_cell(

row, col

) # getting the status of emptying a cell

if deleted\_status:

empty\_no\_of\_cell -= 1

coordinates.append((row, col)) # add the coordinates

empty\_cells\_in\_cols = self.\_\_check\_if\_all\_columns\_have\_empty\_cell()

return coordinates

def make\_grid(self) -> list:

"""

This function creates a grid of random numbers in the range of 10 to 99.

It iterates over the number of rows and columns in the grid and appends a random number to each cell.

The function returns the grid as a list of lists.

"""

for \_ in range(self.rows):

row = []

for \_ in range(self.cols):

row.append(

self.r.generate\_random\_number(10, 99)

) # add the random number generated to the list

self.grid.append(row) # add the random number list generated to the list

return self.grid

def create\_table(self) -> PrettyTable:

"""

This function creates a table using PrettyTable

Args:

None

Returns:

PrettyTable: a table with all of the data in the 2D grid `self.grid`

"""

table = PrettyTable(header=False) # Create a table

for row in self.grid:

table.add\_row(row) # Add a row

return table

def generate\_string(

self,

) -> str:

"""

This function generates a string representation of the grid.

Returns:

str: A string representation of the grid.

"""

return str(self.create\_table().get\_string()) # Return a string without the header

def generate\_html(self):

"""

This function generates an HTML representation of the grid

Returns:

str: An HTML representation of the grid.

"""

return str(self.create\_table().get\_html\_string()) # Return the string without the header

def grid\_maker(self) -> list:

self.make\_grid() # Make a grid

self.generate\_cells\_to\_empty() # Empty random grid

return self.grid # send the final grid

## 4.3 Precolation/helper\_functions.py

from Percolation import random, PrettyTable, os

class Random:

@staticmethod

def generate\_random\_number(start: int, end: int) -> int:

"""

Generate a random integer number between start and end (inclusive).

Args:

start (int): The minimum value of the random integer.

end (int): The maximum value of the random integer (inclusive).

Returns:

int: A random integer number between start and end.

"""

return random.randint(

start, end

) # Create a random integer number between start and end

@staticmethod

def select\_choice(choices: list, probability: tuple[float]) -> list:

"""

Select a random choice from a list based on a given probability distribution.

Args:

choices (list): A list of possible choices.

probability (tuple[float]): A tuple of probabilities corresponding to each choice in `choices`. The sum of all probabilities must be equal to 1.

Returns:

list: A single randomly selected choice from `choices` based on the given probability distribution.

Raises:

ValueError: If the length of `choices` and `probability` do not match.

"""

if len(choices) == len(probability):

return random.choices(

choices, weights=probability

) # Return a random choice from the given probability distribution

raise ValueError(

f"The list `choices` and tuple `probability` must have the same length. {choices} ({len(choices)}) != {probability} ({len(probability)})"

) # Create a error if the length of choices and length of probability is not equal

class Ok\_or\_not:

"""

This class takes a 2D list as input and determines whether each column of the list contains all unique elements.

"""

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

"""

Args:

grid (list): A 2D list of integers. Each row represents a column of the original list.

"""

self.grid = grid

self.t = PrettyTable(header=False)

self.col\_length = len(self.grid[0])

self.cols\_data = {

i: [] for i in range(self.col\_length)

} # create a dictionary with column index id and with a value as a list

self.filter = lambda cols\_data: [

"NO" if "" in cols\_data[col] else "OK" for col in cols\_data

] # filter the columns to see if there are empty cell in a column

def generate(self) -> list:

"""

This function generates a table showing whether each column of the input list contains all unique elements.

The output is a string containing the table in HTML format.

Returns:

str: The table showing whether each column of the input list contains all unique elements.

"""

for row in self.grid:

row = row.copy()

for ele in row:

idx = row.index(ele)

self.cols\_data[idx].append(

ele

) # Add each element to each column

row[idx] = None

list\_ok\_or\_not = self.filter(self.cols\_data) # Filter all the columns

self.t.add\_row(list\_ok\_or\_not) # Add to the self.t table

return list\_ok\_or\_not

def get\_html(self) -> str:

"""

This function returns the table showing whether each column of the input list contains all unique elements

in HTML format.

Returns:

str: The table showing whether each column of the input list contains all unique elements in HTML format.

"""

return self.t.get\_html\_string(header=False) # get a html output of the table

def get\_string(self) -> str:

"""

This function returns the table showing whether each column of the input list contains all unique elements

in string format.

Returns:

str: The table showing whether each column of the input list contains all unique elements in string format.

"""

return self.t.get\_string(header=False) # get a string output of the table

def grid\_condition(dims: str) -> tuple:

"""

This function takes a string input in the form of "x" separated row and column dimensions and returns a tuple of integers representing the dimensions.

The function ensures that the input is in the correct format and that the row and column dimensions are within the specified range.

If the input is not in the correct format or the dimensions are outside the specified range, the function returns a default value of (5, 5).

Args:

dims (str): A string in the form of "x" separated row and column dimensions.

Returns:

tuple: A tuple of integers representing the row and column dimensions.

"""

split = dims.split(dims[int(len(dims)/2)]) # Splitting the dim by 'middle' character

if any([s.isnumeric() for s in split]): # checking if the picked character is a number

split = dims.split('x') # then try and split from 'x'

if len(split) == 2: # Checking conditions

rows, cols = split

if rows.isnumeric() and cols.isnumeric():

rows, cols = list(map(int, split))

if 3 <= rows <= 9 and 3 <= cols <= 9:

return rows, cols

print('The enetered grid size is invalid... The default 5x5 Grid is used')

return 5, 5 # Returning the default grid sizes

def director\_creator(directory: str) -> bool:

"""

Creates a directory with the given name in the current working directory.

Parameters:

directory (str): The name of the directory to create.

Returns:

bool: True if the directory was created, False if the directory already exists.

"""

if not os.path.isdir(f"./{directory}"): # checking if the directory doesnt exist

os.mkdir(f"./{directory}") # making the directory

return True

return False

## 4.4 Precolation/html.py

from Percolation import Grid\_Maker, Ok\_or\_not, director\_creator

class HTML:

def \_\_init\_\_(self, grid: Grid\_Maker, ok\_or\_not: Ok\_or\_not, file\_name: str) -> None:

"""

Initialize the HTML class.

Args:

grid (Grid\_Maker): The grid maker object.

ok\_or\_not (Ok\_or\_not): The ok or not object.

file\_name (str): The file name.

"""

self.grid = grid

self.ok\_or\_not = ok\_or\_not

self.file\_name = f"./{file\_name}/{file\_name}.html" # the directory

# calling the director\_creator() helper function

director\_creator(file\_name)

@staticmethod

def create\_html\_code(elements: str, name: str="Grid") -> str:

return f"""

<!DOCTYPE html>

<html lang="en">

<head>

<title>{name}</title>

</head>

<body>{elements}</body>

</html>

""" # return the default html code template

def create\_full\_grid(self) -> str:

"""

This function generates the full HTML code for the Percolation simulation.

Returns:

str: The full HTML code for the Percolation simulation.

"""

return (

self.grid.generate\_html() + self.ok\_or\_not.get\_html()

) # returning the combination of grid and ok or not

def create\_file(self) -> str:

"""

This function generates the full HTML code for the Percolation simulation.

Returns:

str: The full HTML code for the Percolation simulation.

"""

code = self.create\_html\_code(

self.create\_full\_grid()

) # generating the html code

with open(self.file\_name, "w") as f:

f.write(code) # inserting the code into the file

return code

## 4.5 Precolation/txt.py

from Percolation import Grid\_Maker, Ok\_or\_not, director\_creator

class Text:

def \_\_init\_\_(self, grid: Grid\_Maker, ok\_or\_not: Ok\_or\_not, file\_name: str) -> None:

"""

Initialize a new instance of the Text class.

Args:

grid (Grid\_Maker): an instance of the Grid\_Maker class that generates the text grid

ok\_or\_not (Ok\_or\_not): an instance of the Ok\_or\_not class that generates the "ok" or "not ok" message

file\_name (str): the name of the file to be created

"""

self.grid = grid

self.ok\_or\_not = ok\_or\_not

self.file\_name = f"./{file\_name}/{file\_name}.txt"

director\_creator(file\_name)

def create\_full\_grid(self) -> str:

"""

This function combines the output of the grid and the ok\_or\_not functions to create a full text grid.

Returns:

str: the combined output of the grid and the ok\_or\_not functions

"""

return (

self.grid.generate\_string() + "\n" + self.ok\_or\_not.get\_string()

) # create the string

def create\_file(self) -> str:

"""

This function creates a text file with the combined output of the grid and the ok\_or\_not functions.

Returns:

str: the full text grid with the "ok" or "not ok" message

"""

txt = self.create\_full\_grid() # getting the full string of the grid

with open(self.file\_name, "w") as f:

f.write(txt) # add to the file

return txt

## 4.6 main.py

from Percolation import grid\_condition, Grid\_Maker, Ok\_or\_not, HTML, Text, sys, datetime

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

rows, cols = grid\_condition(sys.argv[-1])

gm = Grid\_Maker(rows, cols)

two\_dim\_grid = gm.grid\_maker()

oon = Ok\_or\_not(two\_dim\_grid)

ok\_or\_not\_list = oon.generate()

file\_name = datetime.datetime.now().strftime("%Y\_%m\_%d\_%H%M")

HTML(gm, oon, file\_name).create\_file()

t = Text(gm, oon, file\_name).create\_file()

print(t)

# 5 Test Cases

## 5.1 Valid Grid

The following test case is the valid grid.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Command | Input Entered | Expected  Outcome | Actual Outcome | Results |
| python3 main.py  4x4 | 4x4 | Display a 4x4  Grid | Display a 4x4  Grid | Pass |

A black screen with white text

Description automatically generated

Figure 26

## 5.2 No Command Line Arguments (Default)

The following test case is the default grid.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Command | Input Entered | Expected  Outcome | Actual Outcome | Results |
| python3 main.py |  | Display a  Default 5x5 Grid | Display a  Default 5x5 Grid | Pass |

A screenshot of a computer program

Description automatically generated

Figure 27

## 5.3 Invalid Grid

The following test case is the output for a invalid grid size, an error message and the use of the default 5x5 Grid.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Command | Input Entered | Expected  Outcome | Actual Outcome | Results |
| python3 main.py  10x3 | 10x3 | Display a error message and  Create a Default  5x5 Grid | “The enetered grid size is invalid... The default 5x5 Grid is used” &  Create a 5X5  Grid | Pass |

A computer screen with white text

Description automatically generated

Figure 28

## 5.4 Impossible Percolation

The following test case is the output for impossible percolation, where each column will have a empty cell and the percolation status for each column is “NO”.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Command | Input Entered | Expected  Outcome | Actual Outcome | Results |
| python3 main.py  3x3 | 3x3 | Display a 3x3  Grid with a Precolation  Status of all  “NO” | Display a 3x3  Grid with a Precolation  Status of all  “NO” | Pass |

A black screen with white text

Description automatically generated

Figure 29

## 5.5 Minimum Grid Dimensions

The minimum grid size that percolation is simulated of.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Command | Input Entered | Expected  Outcome | Actual Outcome | Results |
| python3 main.py  3x3 | 3x3 | Display a 3x3  Grid | Display a 3x3  Grid | Pass |

A black screen with white text

Description automatically generated

Figure 30

## 5.6 Maximum Grid Dimensions

The maximum grid size that percolation is simulated of.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Command | Input Entered | Expected  Outcome | Actual Outcome | Results |
| python3 main.py  9x9 | 9x9 | Display a 9x9  Grid | Display a 9x9  Grid | Pass |

A screenshot of a computer

Description automatically generated

Figure 31

## 5.7 Text File Generation

A Text File being created.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Command | Input Entered | Expected  Outcome | Actual Outcome | Results |
| python3 main.py  6x5 | 6x5 | Display a 6x5 Grid and Create a text file with the 6x5 | Display a 6x5 Grid and Create a text file with the 6x5 | Pass |

A screenshot of a computer

Description automatically generated

Figure 32

## 5.8 HTML File Generation

A HTML File being created.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Command | Input Entered | Expected  Outcome | Actual Outcome | Results |
| python3 main.py  6x5 | 6x5 | Display a 6x5 Grid and Create a html file with  the 6x5 | Display a 6x5 Grid and Create a html file with  the 6x5 | Pass |

A screenshot of a computer

Description automatically generated

Figure 33

# 6 Future Improvements

The current percolation simulation program has implemented a basic level implementation, the program has been built with scalability, maintenance, and future implementation in mind. The following are a few possible future features that could be implemented to the current program:

* Enhanced Visualization: Create more advanced visualizations using libraries such as Matplotlib or Plotly. This would offer a better way for users to engage with the results.
* Parameterization: Allowing users to add more console arguments to adjust more parameters such as percentage (%) of empty cells, range of random numbers.
* Export to Other Formats: Adding the ability to export results to more formats such as CSV or Excel for further analysis.

# 7 Conclusion

In conclusion, the Python program developed achieved the objectives set out of simulating the percolation process. Grids are generated with random two digital numbers and random empty cells, the program mimics real world scenarios of liquid passing through filters. The program can produce HTML and Text Files as well. Overall, the program is a useful tool to study and understand percolation.