



## Foundation Certificate in Higher Education

**Module:** DOC 334 Introduction to Programming in Python – P2

**Module Leader:** Mr. Nishan Saliya Harankahawa

**Assignment Number:** 1

**Assignment Type:** Individual

**Issue Date:** 2022-11-18

**Hand – in – Date:** 2022-12-18

**Deadline:** on or before 11:59 PM (2022-12-18)

**Qualifying mark:** 40%

**Student Name:** Shrena Kishokumar

**Student ID:** 20220283

## **I. Abstract**

This report consists of the explanation of a simple percolation program which automatically creates a table as per the user input of rows and columns and identifies if percolation is possible. It speaks briefly of the problem which must be solved, and elaborately about the solution to how this assignment was handled and programmed. The test cases done to test the working of the program and the program code are mentioned in this document for reference.

## **II. Acknowledgments**

I am beyond thankful to our Module Leader, Mr. Nishan Saliya Harankahawa for helping me understand the basic concepts of programming and providing the required knowledge, also assisting towards the completion of this assignment.

I also thank, Ms. Jananie Mayooreesan for providing the knowledge on how a report must be written and to handle MS Word.

# Table of Contents

I. Abstract.....	ii
II. Acknowledgments .....	iii
III. List of Figures .....	v
IV. List of Tables .....	v
1. Problem .....	1
2. Solution .....	1
2.1 Recognizing Number of Rows and Columns .....	1
2.2 Generating Random Two Digit Numbers and Spaces for the Grid.....	2
2.3 Performing Percolation and Displaying OK, NO.....	2
2.4 Displaying Result on a Webpage .....	3
2.5 Displaying the Result on .txt File.....	3
3. Test Case .....	4
3.1 Test Cases Expected and Result.....	4
3.2 Test Cases Evidences .....	5
Test Case #1.....	5
Test Case #2.....	6
Test Case #3.....	6
Test Case #4.....	6
Test Case #5.....	6
Test Case #6.....	7
Test Case #7.....	7
Test Case #8.....	8
Test Case #9.....	9
Test Case #10.....	9
4. Program Codes.....	11

### III. List of Figures

Figure 1: Test Case 1 Console Display .....	5
Figure 2: Test Case 1 Text File Display .....	5
Figure 3: Test Case 1 Webpage Display .....	5
Figure 4: Test Case 2 Console Display .....	6
Figure 5: Test Case 3 Console Display .....	6
Figure 6: Test Case 4 Console Display .....	6
Figure 7: Test Case 5 Console Display .....	6
Figure 8: Test Case 6 Console Display .....	7
Figure 9: Test Case 6 Text File Display .....	7
Figure 10: Test Case Webpage Display .....	7
Figure 11: Test Case 7 Console Display .....	7
Figure 12: Test Case 8 Console Display .....	8
Figure 13: Test Case 8 Text File Display .....	8
Figure 14: Test Case 8 Webpage Display .....	8
Figure 15: Test Case 9 Console Display .....	9
Figure 16: Test Case 10 Console Display .....	9
Figure 17: Test Case 10 Text File Display .....	9
Figure 18: Test Case 10 Webpage Display .....	10
Figure 19: Created Text Files and HTML Files.....	10

### IV. List of Tables

Table 1: Test Case for Percolation.....	4
---	---

# 1. Problem

The problem is to create a program which runs with the command `perc` in the command console to do percolation process. The users are able to input number of rows and columns as preferred, as long as it is in the range of 3x3 and 9x9. If no user input is given, the grid is to be taken in a 5x5 size. The program must randomize two digit numbers and blank spaces and input it into the grid, then further must check each column to see if all rows of the particular column consist of number, if yes, the message “OK” must be displayed below the particular column and if no, the message “NO” must be displayed below the column.

Furthermore, once the program finishes running, the output table must be printed on to a text file and an html file to view later. The text files and html files must not be overwritten, but instead every time the code is run, a new file must be written.

# 2. Solution

## 2.1 Recognizing Number of Rows and Columns

The solution found to this problem is first locating the number of rows and columns the grid must consist of. By importing **sys module** into python and using the `sys.argv[]` function, if the user inputs a particular value, the program checks to see if the value exceeds the number of inputs that can be entered (for example: 11x3 cannot be entered) by checking the length of `sys.argv[1]`. When the conditions are fulfilled, the program continues to collect the input of the user as a string, then separates the elements in the string and appends it to a list. By using this method, we can easily assign the `list[0]` and `list[2]` to row and column variables respectively.

## 2.2 Generating Random Two Digit Numbers and Spaces for the Grid

The next step of the program is to randomize numbers and spaces which will be input into the table. To start off this step, the **random module** is imported into Python. Then, a random number generator (rand variable) with a range of 500, generates a number. The program checks if the number generated is divisible by 2 (with no remainder), and if it is an even number, a second random number generator (rand2 variable) with the range from 10 to 99 (two digit numbers), produces a number and that is appended to a list, but if the first generated number is not an even number, then a “\_\_” is appended to the list.

This process is done until the list consists of the same number of elements same number of elements as the total number of elements in the table (total elements = row\*column).

Then the list was broken down to smaller lists to create the rows of the table. So while the length of the list was greater than or equal to the number of columns in the table, the list is divided by the number of rows so it gets split with elements to fill in columns of each row. Then the list gets added to the table as a row.

Finally using the **PrettyTables module**, the tables rules are set and the table (table 1) is printed to display.

## 2.3 Performing Percolation and Displaying OK, NO

The percolation is done by checking each column of table 1 to see if “\_\_” exists as an input. This is done by appending the values to a list in a column by changing the number of row (by changing the value of n in [n][m] such that [0][0] , [1][0], [2][0]), then once the row ends, the program checks if a blank is in present in that list elements. If any of the elements is a “\_\_”, then the answer “NO” is appended to another list, while the list with the elements of the column gets cleared. Then [m] gets added by 1, so that the elements of the next column is added to the list to be checked for percolation. This process is repeated until m is equal to the number of columns.

Then the list containing the “OK” and “NO” is then entered into another table (table 2) created using PrettyTable with no borders (since modifying borders for only specific rows was not available in the module), to be displayed right under the first table.

## 2.4 Displaying Result on a Webpage

The result is to be printed on a webpage with borders around the entire table, thus the “OK”, “NO” list was added to table 1 as a row to easily display it on the webpage.

The function `get_html_string` in `PrettyTable` module allows you to get the HTML code to display the result, so that was assigned to the variable `ht0` with specific attributes so the display will be in table format.

Using the **Time module** imported into Python, rounding off time and multiplying it by the value 1000 was assigned to variable `s`, which can help to create a different file every time the code is run. So using the method to open a new file with the name “Web –” and then the number given from the time, and ending the file name with “.html”, the program is able to produce an html file with a different name every time. The `ht0` variable with the HTML code is then written into the html file, providing an output in the webpage.

## 2.5 Displaying the Result on .txt File

The previously added row for table 1, when printed onto .txt file was displayed with improper alignment. Thus, it was deleted, so that table 1 and table 2 were printed separately into the .txt file. The rules of table 1 were modified to fit the display of a text file, since borders cannot be printed into the file. Using the same `s` variable used in the above scenario, the file name “Table –”, number from `s`, ending the file name with “.txt” was opened. Then table 1 was first added, as a string, because .txt files do not support `PrettyTable` elements, then after a break of line, table 2 was added to the text file.



### 3. Test Case

#### 3.1 Test Cases Expected and Result

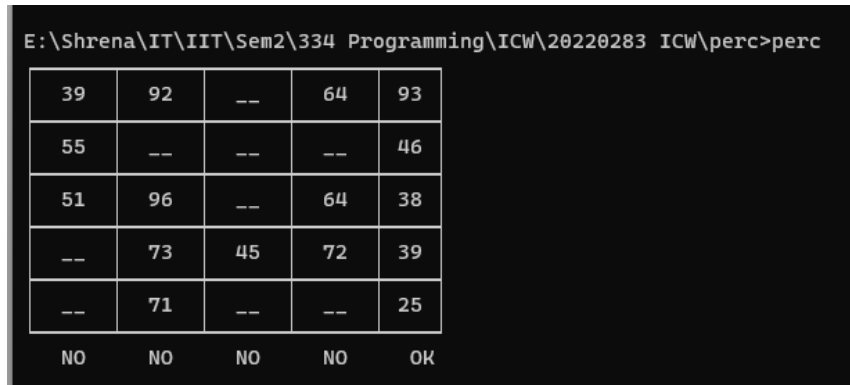
Test Case #	Inputs	Expected Outcome	Actual Outcome	Remarks
1	perc	5x5 grid with randomized numbers, and successful percolation Text File with result Webpage with result	5x5 grid with randomized numbers, and successful percolation Text File with result Webpage with result	Test Case Passed
2	perc 1x5	“Invalid Number of Rows”	“Invalid Number of Rows”	Test Case Passed
3	perc 4x2	“Invalid Number of Columns”	“Invalid Number of Columns”	Test Case Passed
4	perc 8g8	“Invalid Syntax”	“Invalid Syntax”	Test Case Passed
5	perc 4x33	“Invalid Grid Size”	“Invalid Grid Size”	Test Case Passed
6	perc 3x3	3x3 grid with randomized numbers, and successful percolation Text File with result Webpage with result	3x3 grid with randomized numbers, and successful percolation Text File with result Webpage with result	Test Case Passed
7	perc 444	“Invalid Syntax”	“Invalid Syntax”	Test Case Passed
8	perc 5x8	5x8 grid with randomized numbers, and successful percolation Text File with result Webpage with result	5x8 grid with randomized numbers, and successful percolation Text File with result Webpage with result	Test Case Passed
9	perc 8x	“Incomplete Grid Size”	“Incomplete Grid Size”	Test Case Passed
10	perc 6x3	6x3 grid with randomized numbers, and successful percolation Text File with result Webpage with result	6x3 grid with randomized numbers, and successful percolation Text File with result Webpage with result	Test Case Passed

Table 1: Test Case for Percolation

## 3.2 Test Cases Evidences

### Test Case #1

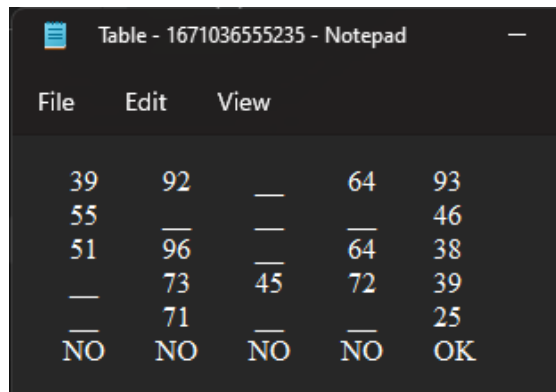
When no user input is given for the program to run, by default it creates a 5x5 table as below:



The screenshot shows a command prompt window with the command `E:\Shrena\IT\IIT\Sem2\334 Programming\ICW\20220283 ICW\perc>perc`. Below the command, a 5x5 table is displayed. The first four rows contain numbers and dashes, while the fifth row contains the status messages 'NO', 'NO', 'NO', 'NO', and 'OK'.

39	92	--	64	93
55	--	--	--	46
51	96	--	64	38
--	73	45	72	39
--	71	--	--	25
NO	NO	NO	NO	OK

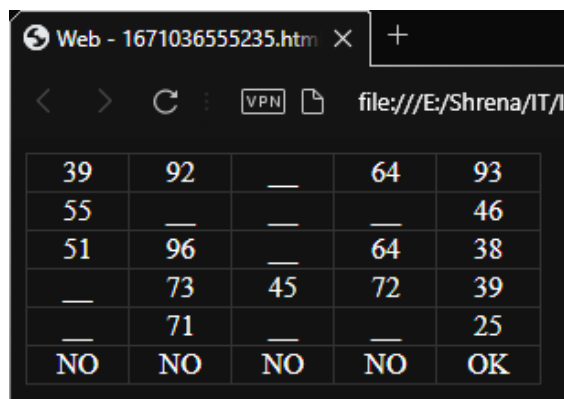
Figure 1: Test Case 1 Console Display



The screenshot shows a Notepad window titled 'Table - 1671036555235 - Notepad'. The window contains the same 5x5 table as seen in the console display.

39	92	--	64	93
55	--	--	--	46
51	96	--	64	38
--	73	45	72	39
--	71	--	--	25
NO	NO	NO	NO	OK

Figure 2: Test Case 1 Text File Display



The screenshot shows a web browser window with the address bar displaying `file:///E:/Shrena/IT/I`. The browser displays the same 5x5 table as seen in the previous figures.

39	92	--	64	93
55	--	--	--	46
51	96	--	64	38
--	73	45	72	39
--	71	--	--	25
NO	NO	NO	NO	OK

Figure 3: Test Case 1 Webpage Display

## Test Case #2

When the input for row is lesser than 3, it shows an error such as:

```
E:\Shrena\IT\IIT\Sem2\334 Programming\ICW\20220283 ICW\perc>perc 1x5  
Invalid Number of Rows  
E:\Shrena\IT\IIT\Sem2\334 Programming\ICW\20220283 ICW\perc>|
```

*Figure 4: Test Case 2 Console Display*

## Test Case #3

When the input for column is lesser than 3, it shows an error such as:

```
E:\Shrena\IT\IIT\Sem2\334 Programming\ICW\20220283 ICW\perc>perc 4x2  
Invalid Number of Columns  
E:\Shrena\IT\IIT\Sem2\334 Programming\ICW\20220283 ICW\perc>|
```

*Figure 5: Test Case 3 Console Display*

## Test Case #4

When the syntax between row and column is anything other than “x”, it shows an error such as:

```
E:\Shrena\IT\IIT\Sem2\334 Programming\ICW\20220283 ICW\perc>perc 8g8  
Invalid Syntax  
E:\Shrena\IT\IIT\Sem2\334 Programming\ICW\20220283 ICW\perc>|
```

*Figure 6: Test Case 4 Console Display*

## Test Case #5

When the user input exceeds 3 characters, it shows an error such as:

```
E:\Shrena\IT\IIT\Sem2\334 Programming\ICW\20220283 ICW\perc>perc 4x33  
Invalid Grid Size  
E:\Shrena\IT\IIT\Sem2\334 Programming\ICW\20220283 ICW\perc>|
```

*Figure 7: Test Case 5 Console Display*

## Test Case #6

When perc 3x3 is inputted, the following results are received:

```
E:\Shrena\IT\IIT\Sem2\334 Programming\ICW\20220283 ICW\perc>perc 3x3
```

--	94	18
--	--	88
48	74	84
NO	NO	OK

```
E:\Shrena\IT\IIT\Sem2\334 Programming\ICW\20220283 ICW\perc>
```

Figure 8: Test Case 6 Console Display

Table - 1671037733068 - Notepad

File Edit View

—	94	18
—	—	88
48	74	84
NO	NO	OK

Figure 9: Test Case 6 Text File Display

Web - 1671037733068.htm x +

< > C VPN file:///E

—	94	18
—	—	88
48	74	84
NO	NO	OK

Figure 10: Test Case Webpage Display

## Test Case #7

When perc 444 is inputted, the following error is shown:

```
E:\Shrena\IT\IIT\Sem2\334 Programming\ICW\20220283 ICW\perc>perc 444
Invalid Syntax
```

Figure 11: Test Case 7 Console Display

## Test Case #8

When perc 5x8 is inputted, the following results are received:

E:\Shrena\IT\IIT\Sem2\334 Programming\ICW\20220283 ICW\perc>perc 5x8

66	--	70	83	48	37	--	--
21	--	63	59	19	39	92	31
--	--	24	98	--	33	38	66
--	--	--	63	26	60	98	54
--	--	--	98	22	--	--	--
NO	NO	NO	OK	NO	NO	NO	NO

E:\Shrena\IT\IIT\Sem2\334 Programming\ICW\20220283 ICW\perc>|

Figure 12: Test Case 8 Console Display

Table - 1671038073326 - Notepad

File Edit View

66	--	70	83	48	37	--	--
21	--	63	59	19	39	92	31
--	--	24	98	--	33	38	66
--	--	--	63	26	60	98	54
--	--	--	98	22	--	--	--
NO	NO	NO	OK	NO	NO	NO	NO

Figure 13: Test Case 8 Text File Display

Web - 1671038073326.htm X +

< > ↻ VPN file:///E:/Shrena/IT/IIT/Sem2/334%20Programming/10

66	--	70	83	48	37	--	--
21	--	63	59	19	39	92	31
--	--	24	98	--	33	38	66
--	--	--	63	26	60	98	54
--	--	--	98	22	--	--	--
NO	NO	NO	OK	NO	NO	NO	NO

Figure 14: Test Case 8 Webpage Display

## Test Case #9

When a grid size is not completed, the following error is shown:

```
E:\Shrena\IT\IIT\Sem2\334 Programming\ICW\20220283 ICW\perc>perc 8x
Incomplete Grid Size
```

Figure 15: Test Case 9 Console Display

## Test Case #10

When perc 6x3 is inputted, following outputs are received:

```
E:\Shrena\IT\IIT\Sem2\334 Programming\ICW\20220283 ICW\perc>perc 6x3
```

49	--	47
--	16	64
--	--	59
--	--	42
--	65	54
--	--	95
NO	NO	OK

Figure 16: Test Case 10 Console Display

Table - 1671038444135 - Notepad

File	Edit	View
49	—	47
—	16	64
—	—	59
—	—	42
—	65	54
—	—	95
NO	NO	OK

Figure 17: Test Case 10 Text File Display

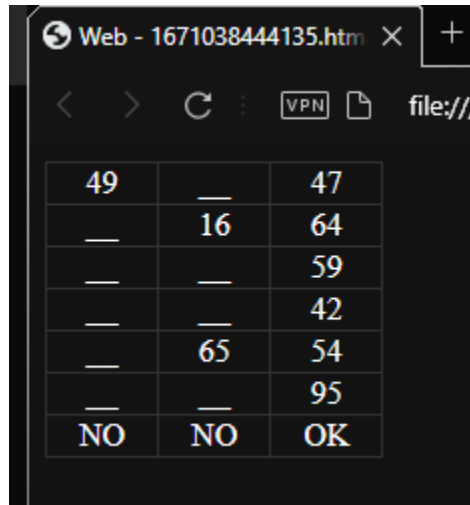


Figure 18: Test Case 10 Webpage Display

Name	Date modified	Type	Size
perc	12/14/2022 10:48 PM	Python File	5 KB
Table - 1671036555235	12/14/2022 10:19 PM	Text Document	1 KB
Table - 1671038073326	12/14/2022 10:44 PM	Text Document	1 KB
Table - 1671038444135	12/14/2022 10:50 PM	Text Document	1 KB
Web - 1671036555235	12/14/2022 10:19 PM	Opera GX Web Do...	4 KB
Web - 1671038073326	12/14/2022 10:44 PM	Opera GX Web Do...	6 KB
Web - 1671038444135	12/14/2022 10:50 PM	Opera GX Web Do...	3 KB

Figure 19: Created Text Files and HTML Files

## 4. Program Codes

```
#importing sys, time, random and prettytable modules

import sys

import random

from time import time

from prettytable import PrettyTable as pt, SINGLE_BORDER, ALL


#defining functions

def table_rule():

    "The function to set table rules for tb1 and tb2"

    tb1.header = False

    tb1.hrules = ALL

    tb1.set_style(SINGLE_BORDER)


    tb2.header = False

    tb2.set_style(SINGLE_BORDER)

    tb2.border = False

    tb2.left_padding_width = 3

    tb2.right_padding_width = 2


def table_rule_txt():

    "This function sets the table rules to make it printable on txt file"

    tb1.vertical_char = ''
```



```
tb1.horizontal_char = ''
tb1.junction_char = ''
tb1.border = False
tb1.preserve_internal_border = True
tb1.max_width = 5
tb1.padding_width = 3
tb2.vertical_char = ''
tb2.horizontal_char = ''
tb2.junction_char = ''
tb2.border = False
tb2.preserve_internal_border = True
tb2.max_width = 2
tb2.right_padding_width = 3
```

```
#declaring variables
```

```
lst = [] #to seperate user input
```

```
lst3 = [] #to input random inputs to display in grids
```

```
listt = [] #to add OK and NO
```

```
ans = [] #to append the inputs in each column
```

```
tb1 = pt()
```

```
tb2 = pt()
```

```
i = " __ "
```

```
rand = 0
```

```
rand2 = 0
```

```
row1 = 0
```

```
col = 0
```

```
tot = 0
```

```
gen = 1
```

```
fo = 0
```

```
ht = 0
```

```
n = 0
```

```
m = 0
```

```
p = 1
```

```
#setting table rules
```

```
table_rule()
```

```
#declaring arguments in sys
```

```
if len(sys.argv) > 1:
```

```
    first = sys.argv[1]
```

```
else:
```

```
    first = "5x5"
```

```
if 4 <= len(first):
```

```
    print ("Invalid Grid Size")
```

```

elif len(first) <= 2:

print ("Invalid Grid Size")

else:

for letter in first:

lst.append(letter) #seperating the values to recognize no. of rows and columns


if lst[1] == "x":

#declaring rows and columns

row1 = int(lst[0])

col = int(lst[2])

tot = row1*col


if 3 <= row1 <= 9:    #setting minimum and maximum number of rows and columns

if 3 <= col <= 9:

while gen <= tot:

rand = random.randrange(500)

rand2 = random.randrange(10,99)


if rand%2 == 0: #generates random number and checks if it even

lst3.append (rand2) #if even, two digit number is appended to list

gen += 1

else:

```

```

lst3.append (" __ ") #if odd, blank is appended to list

gen += 1

random.shuffle(lst3)


#seperates one list into many as per the number of rows while adding rows to a table

sublist = len(lst3) // row1

s1, s2 = lst3[:sublist], lst3[sublist:]

tbl.add_row(s1)


while col <= len(s2):

    sublist = len(lst3) // row1

    s3, s2 = s2[:sublist], s2[sublist:]

    tbl.add_row(s3)

print (tbl)


#percolation process done by changing each number of column and rows

while p < tot:

    for row in tbl:

        ans.append(tbl.rows[n][m])

        n+=1

    if row1 <= n:

        if i in ans:

            listt.append("NO")

```

```

else:

listt.append("OK")

ans.clear()

n=0

m += 1

if col <= m:

break


tb2.add_row(listt)

print(tb2)


#getting html string and saving file as html to display on a web page

tb1.add_row(listt)

ht0 = tb1.get_html_string(attributes={"class":"table"}, format=True)


s = str(round(time() * 1000)) #using time to create new files every time code is run

with open("Web - " + s + ".html","a") as ht:

ht.write(str(ht0))

ht.close


#modifying prettytable rules to print into txt file

table_rule_txt()

tb1.del_row(row1)

```

```
with open("Table - " + s + ".txt","a") as fo:
```

```
fo.write(str(tb1))
```

```
fo.write("\n")
```

```
fo.write(str(tb2))
```

```
fo.close
```

```
else:
```

```
print ("Invalid Number of Columns")
```

```
else:
```

```
print ("Invalid Number of Rows")
```

```
else:
```

```
print("Invalid Syntax")
```