INFORMATICS
INSTITUTE OF
TECHNOLOGY

FOUNDATION CERTIFICATE IN HIGHER EDUCATION

Module: DOC 334 Computer Programming

Module Leader: Mr. Nishan Saliya Harankahawa

Type of Assignment: Individual Assignment

Topic: ICW Report

Issue Date: 11-03-2024

Submission Date: 30-03-2024

Student Name: M.A.P.K. Nethsarani

Student ID: 20231336

Abstract

This project aimed to create a Python program that mimics a simple percolation process, similar to how liquids filter through a substance like in coffee-making. The program generates a grid of random two-digit numbers with some empty slots scattered randomly. It then checks each column in the grid to see if percolation can happen. Percolation is possible in a column if it has numbers from top to bottom without any empty spaces. If there are any empty spaces in a column, percolation cannot happen. To achieve this, the program generates the grid, checks each column for percolation, and displays whether percolation is possible or not for each column. This project demonstrates basic Python logic and looping techniques to simulate a simple concept in fluid dynamics.

Acknowledgement

I would like to express my sincere gratitude to Mr. Nishan Saliya Harankahawa, our esteemed module leader for DOC 334 Computer Programming. His guidance, support, and expertise have been invaluable throughout this project, ensuring a deeper understanding of programming concepts and methodologies.

I also extend my thanks to our other lecturers, Mr. Namal Malalasena, Ms. Shafka Fuard, and Ms. Rasheli Nimansha, for their contributions to our learning journey in computer programming. Their dedication and commitment to imparting knowledge have enriched our understanding of the subject and inspired us to explore new horizons in programming.

Finally, I acknowledge the support of my family and friends, whose encouragement have been a constant source of motivation throughout this endeavor.

Table of Contents

List of Figures

List of Tables

1. Full Python Code

```
# Import necessary modules

import random

import datetime

from prettytable import PrettyTable as PT

import sys



# Main function to execute the program

def main():

    # Default dimensions

    num_rows = 5

    num_cols = 5



    # Check if user input is provided

    if len(sys.argv) > 1:

        user_input = sys.argv[1]

        if 'x' in user_input:

            dimensions = user_input.split('x')

            if len(dimensions) == 2:

                try:
```

```
            num_rows = int(dimensions[0])
            num_cols = int(dimensions[1])
            if not (3 <= num_rows <= 9 and 3 <= num_cols <= 9):
                print("Error: Dimensions must be between '3x3' and '9x9'.")
                return
        except ValueError:
            print("Error: Invalid dimensions format.")
            return
    else:
        print("Error: Invalid dimensions format.")
        return
else:
    try:
        num_rows, num_cols = map(int, user_input.split())
        if not (3 <= num_rows <= 9 and 3 <= num_cols <= 9):
            print("Error: Dimensions must be between '3 3' and '9 9'.")
            return
    except ValueError:
        print("Error: Invalid input format.")
        return
    print(f"Valid dimensions: {num_rows}x{num_cols}")

    # Generate matrix based on user-specified or default dimensions
    matrix = generate_matrix(num_rows, num_cols)
    # Display the generated matrix
    display_matrix(matrix)
    # Check percolation in the matrix
    print(check_percolation(matrix))
    # Save the matrix and percolation data
    save_matrix(matrix)


# Function to generate a matrix with random values and empty cells
def generate_matrix(rows, cols, empty_prob=0.2):
    matrix = []
    for _ in range(rows):
        row = []
        for _ in range(cols):
            # Randomly decide whether to add a number or leave cell empty
            if random.random() > empty_prob:
                row.append(random.randint(10, 99))  # Add a random number
            else:
                row.append('')  # Add an empty cell
        matrix.append(row)
    return matrix

# Function to display the matrix using PrettyTable
def display_matrix(matrix):
    table = PT()
    table.header = False
    table.hrules = True
    table.vrules = True
    table.border = True
    for row in matrix:
        table.add_row(row)
    print(table)


# Function to check percolation in the matrix column-wise
def check_percolation(matrix):
    num_cols = len(matrix[0])
    statuses = []
    for col in range(num_cols):
        percolates = True
        for row in matrix:
            # Check if cell in the current column is empty
            if row[col] == '':
                percolates = False  # Matrix doesn't percolate in this column
                break
        statuses.append("OK" if percolates else "NO")  # Add status for the column

    # Format the statuses for display
    max_status_width = max(map(len, statuses))
    formatted_statuses = [f"{status:{max_status_width}}" for status in statuses]
    return " ".join(formatted_statuses)


# Function to save the matrix and percolation data to files
def save_matrix(matrix):
    current_date_time = datetime.datetime.now().strftime("%Y_%m_%d_%H%M")
    text_file_name = f"{current_date_time}.txt"  # Generate unique text file name
    html_file_name = f"{current_date_time}.html"  # Generate unique HTML file name
    with open(text_file_name, "w") as txt_file, open(html_file_name, "w") as html_file:

        html_file.write("<pre>\n")

        for row in matrix:

            txt_file.write(" ".join(str(cell) for cell in row) + "\n")  # Write matrix to text file

            html_file.write(" ".join(str(cell) for cell in row) + "<br>\n")  # Write matrix to HTML file

        txt_file.write(check_percolation(matrix))  # Write percolation data to text file

        html_file.write(check_percolation(matrix).replace(" ", "&nbsp;"))  # Write percolation data to HTML file

        html_file.write("\n</pre>")


    print("Matrix and percolation data saved as", text_file_name, "and", html_file_name)



# Entry point of the program


main()
```

2. Algorithm

2.1Import necessary modules

- random: Used for generating random numbers and deciding whether to add a number or an empty cell to the matrix.
- datetime: Utilized to get the current date and time for generating unique file names when saving data.
- PrettyTable as PT: Imported to format and display the matrix nicely.
- sys: Used to access command-line arguments (sys.argv) for specifying matrix dimensions.

2.2 Main () function:

- Sets default dimensions for the matrix (5x5).
- Checks if user input is provided via command-line arguments (sys.argv) and updates the dimensions accordingly.

2.3Parsing User Input

- Checks command-line arguments for user input specifying matrix dimensions (sys.argv[1]).
- Parses dimensions in the format “NxM” or “N M” where ‘N’ and ‘M’ are integers representing rows and columns, respectively.
- Ensures that the dimensions are within the range of “3x3” to “9x9”.

2.4 Define the generate_matrix() function

- Generates a matrix based on specified or default dimensions.
- Each cell in the matrix is randomly assigned a number (10-99) or left empty based on a probability (empty_prob).

2.5 Define the display_matrix() function

- Uses “PrettyTable” to format and display the matrix in a tabular format with borders.

2.6 Define the check_percolation() function

- Checks if each column in the matrix has at least one non-empty cell.
- Returns a string indicating whether each column “OK” (percolates) or “NO” (does not percolate).

2.7 Define the save_matrix() function

- Generates unique file names based on the current date and time for saving matrix data to text and HTML files.
- Writes the matrix and percolation data to the respective files.

2.8 Entry point of the program

- Ensures that “main()” is executed when the script is run directly, not when it's imported as a module.

3. Explanation for Each Part

3.1User Input Validation

- The program checks if user input is provided via command-line arguments (sys.argv).
- It validates the input format for dimensions (either ‘NxM’ or ‘N M’ where N and M are integers).
- It ensures that dimensions are within the range of ‘3x3’ to ‘9x9’ (both inclusive).
- If the input is invalid, the program displays an error message and exits.

3.2Matrix Generation (‘generate_matrix’ function)

- Takes input parameters for the number of rows (“rows”), number of columns (“cols”), and optional empty probability (“empty_prob”).
- Creates an empty matrix.
- Iterates over each cell in the matrix and randomly decides whether to add a random number or leave the cell empty based on the “empty_prob”.
- Returns the generated matrix.

3.3Displaying the Matrix (‘display_matrix’ function)

- Takes a matrix as an argument.
- Uses “PrettyTable” to create a table for the matrix display.
- Sets table properties such as header, horizontal rules, vertical rules, and border.
- Adds each row of the matrix to the table.
- Prints the table to display the matrix.

3.4Checking Percolation (‘check_percolation’ function)

- Takes a matrix as an argument.
- Determines percolation column-wise by checking if any cell in each column is empty.

- Stores the percolation status (‘OK’ or ‘NO’) for each column in a list (“statuses”).
- Formats the percolation statuses for display by aligning them based on the maximum status width.
- Returns the formatted percolation statuses as a string.

3.5Saving the Matrix (‘save_matrix’ function)

- Takes a matrix as an argument.
- Generates unique file names for text and HTML files based on the current date and time.
- Writes the matrix and percolation data to both text and HTML files.
- Formats the percolation data for HTML display by replacing spaces with non-breaking spaces (“&nbsp;”).

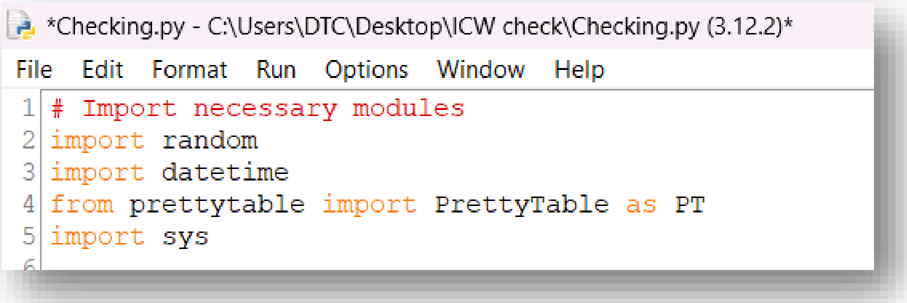
3.6Main Function (‘main’ function)

- Defines default dimensions for the matrix.
- Validates and processes user input for dimensions.
- Displays valid dimensions.
- Generates a matrix based on user-specified or default dimensions using the “generate_matrix” function.
- Displays the generated matrix using the “display_matrix” function.
- Checks percolation in the matrix using the “check_percolation” function and prints the result.
- Saves the matrix and percolation data using the “save_matrix” function.

4. Screenshots

4.1Importing necessary modules

```
# Import necessary modules

import random

import datetime

from prettytable import PrettyTable as PT

import sys
```

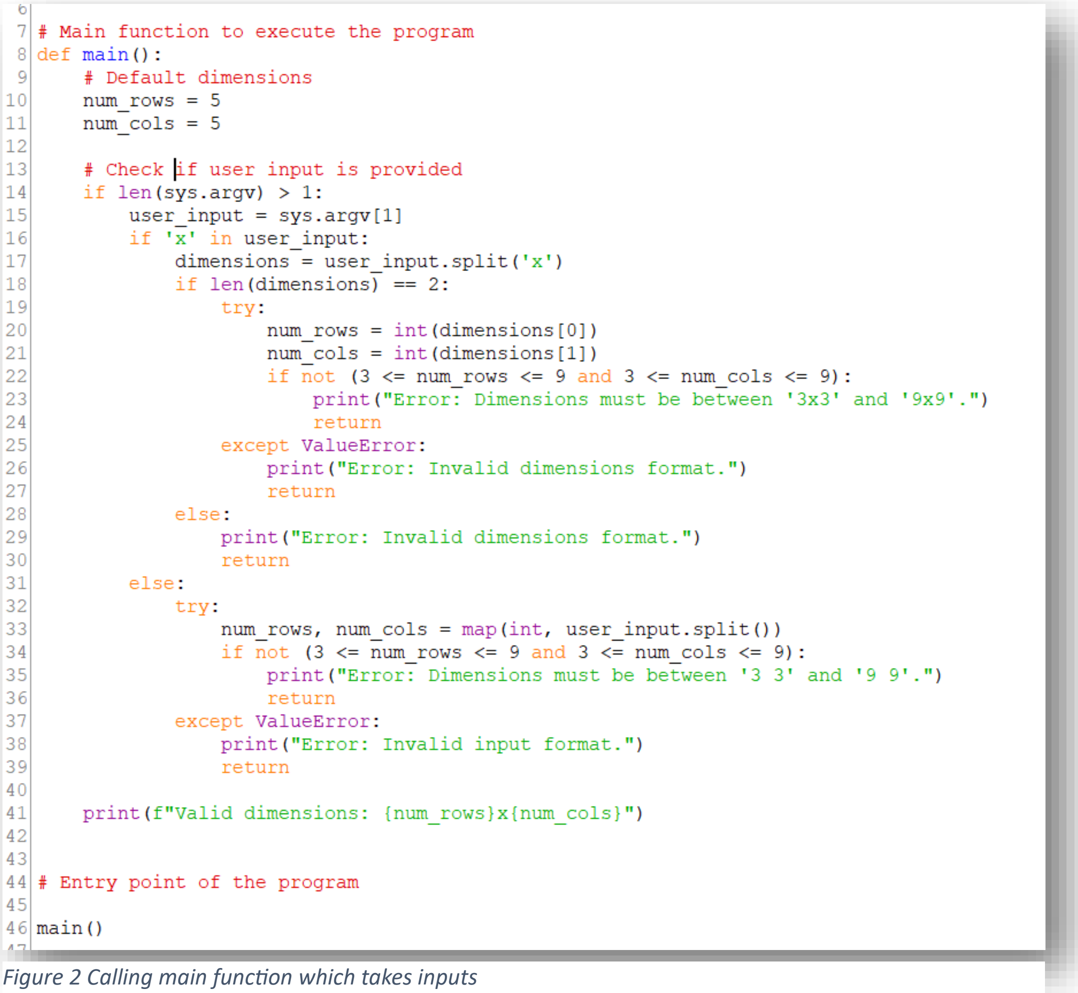

Checking.py - C:\Users\DTC\Desktop\ICW check\Checking.py (3.12.2)
File Edit Format Run Options Window Help

```
# Import necessary modules
import random
import datetime
from prettytable import PrettyTable as PT
import sys
```

Figure 1 Importing modules

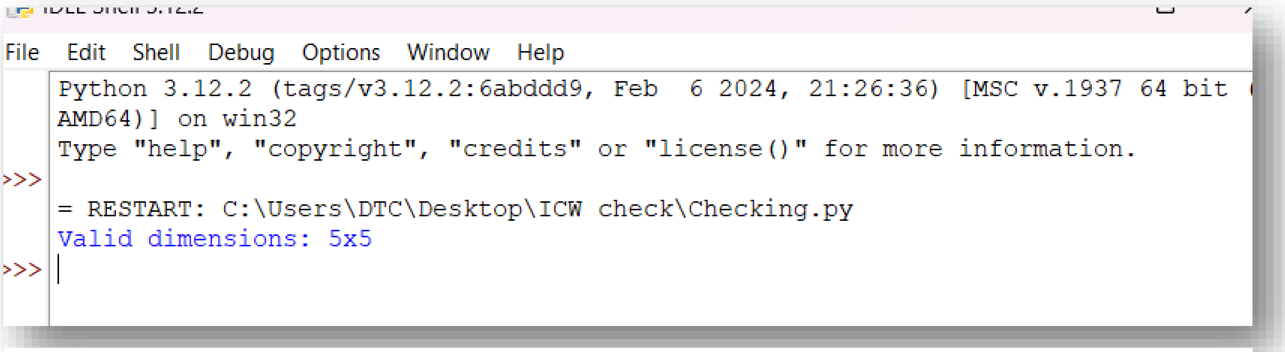
4.2 Calling the main function

```
# Main function to execute the program
def main():
    # Default dimensions
    num_rows = 5
    num_cols = 5

    # Check if user input is provided
    if len(sys.argv) > 1:
        user_input = sys.argv[1]
        if 'x' in user_input:
            dimensions = user_input.split('x')
            if len(dimensions) == 2:
                try:
                    num_rows = int(dimensions[0])
                    num_cols = int(dimensions[1])
                    if not (3 <= num_rows <= 9 and 3 <= num_cols <= 9):
                        print("Error: Dimensions must be between '3x3' and '9x9'.")
                        return
                except ValueError:
                    print("Error: Invalid dimensions format.")
                    return
                else:
                    print("Error: Invalid dimensions format.")
                    return
            else:
                try:
                    num_rows, num_cols = map(int, user_input.split())
                    if not (3 <= num_rows <= 9 and 3 <= num_cols <= 9):
                        print("Error: Dimensions must be between '3 3' and '9 9'.")
                        return
                except ValueError:
                    print("Error: Invalid input format. Please use the format ‘NxM’")
                    return

    print(f"Valid dimensions: {num_rows}x{num_cols}")

# Entry point of the program
main()
```



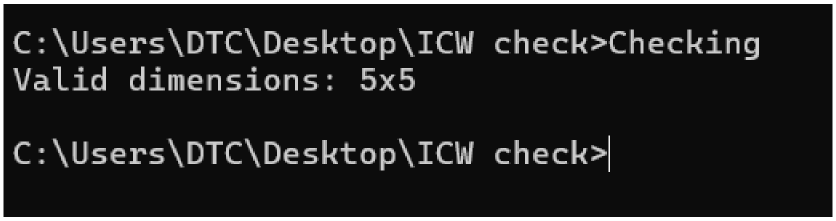
```
# Main function to execute the program
def main():
    # Default dimensions
    num_rows = 5
    num_cols = 5

    # Check if user input is provided
    if len(sys.argv) > 1:
        user_input = sys.argv[1]
        if 'x' in user_input:
            dimensions = user_input.split('x')
            if len(dimensions) == 2:
                try:
                    num_rows = int(dimensions[0])
                    num_cols = int(dimensions[1])
                    if not (3 <= num_rows <= 9 and 3 <= num_cols <= 9):
                        print("Error: Dimensions must be between '3x3' and '9x9'.")
                        return
                except ValueError:
                    print("Error: Invalid dimensions format.")
                    return
            else:
                print("Error: Invalid dimensions format.")
                return
        else:
            try:
                num_rows, num_cols = map(int, user_input.split())
                if not (3 <= num_rows <= 9 and 3 <= num_cols <= 9):
                    print("Error: Dimensions must be between '3 3' and '9 9'.")
                    return
            except ValueError:
                print("Error: Invalid input format.")
                return

    print(f"Valid dimensions: {num_rows}x{num_cols}")


# Entry point of the program

main()
```

Figure 2 Calling main function which takes inputs



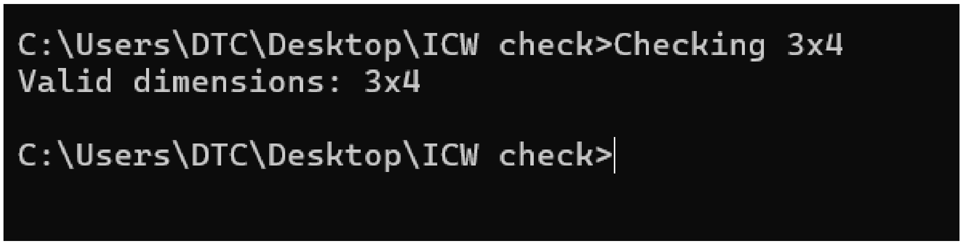
```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\DTC\Desktop\ICW check\Checking.py
Valid dimensions: 5x5
>>>
```

Figure 3 Calling main function which takes inputs results in IDLE



```
C:\Users\DTC\Desktop\ICW check>Checking
Valid dimensions: 5x5

C:\Users\DTC\Desktop\ICW check>
```

Figure 4 calling main function results in command prompt without inputs

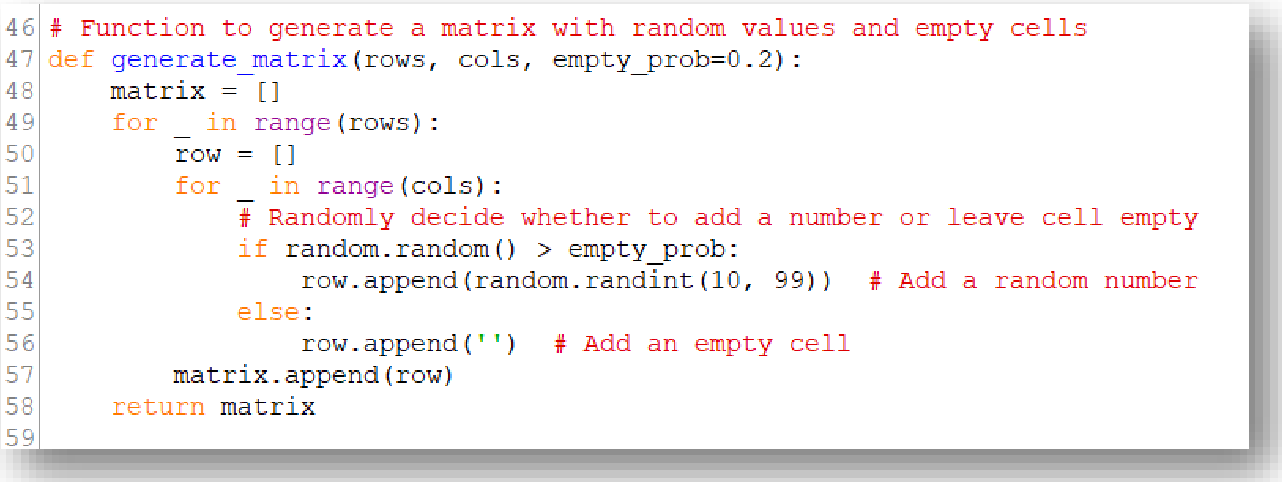


```
C:\Users\DTC\Desktop\ICW check>Checking 3x4
Valid dimensions: 3x4

C:\Users\DTC\Desktop\ICW check>
```

Figure 5 calling main function results in command prompt with inputs

4.3Generation of Matrix

```
# Function to generate a matrix with random values and empty cells

def generate_matrix(rows, cols, empty_prob=0.2):

    matrix = []

    for _ in range(rows):

        row = []

        for _ in range(cols):

            # Randomly decide whether to add a number or leave cell empty
            if random.random() > empty_prob:
                row.append(random.randint(10, 99))  # Add a random number
            else:
                row.append('')  # Add an empty cell
        matrix.append(row)
    return matrix
```

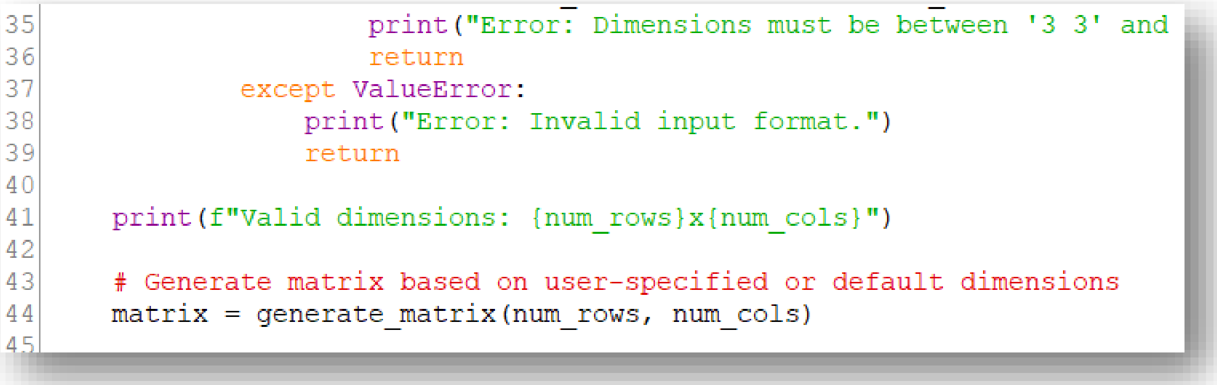


```
# Function to generate a matrix with random values and empty cells
def generate_matrix(rows, cols, empty_prob=0.2):
    matrix = []
    for _ in range(rows):
        row = []
        for _ in range(cols):
            # Randomly decide whether to add a number or leave cell empty
            if random.random() > empty_prob:
                row.append(random.randint(10, 99))  # Add a random number
            else:
                row.append('')  # Add an empty cell
        matrix.append(row)
    return matrix

```

Figure 6 Generate matrix function

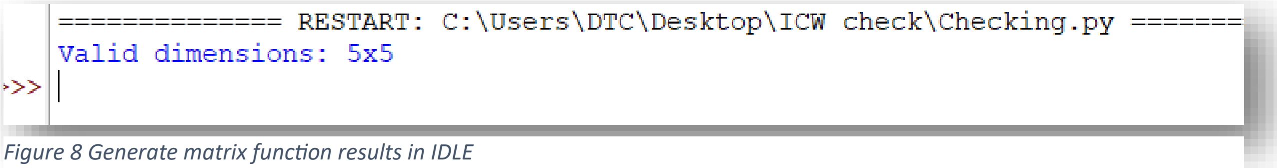
Generate matrix based on user-specified or default dimensions

matrix = generate_matrix(num_rows, num_cols)



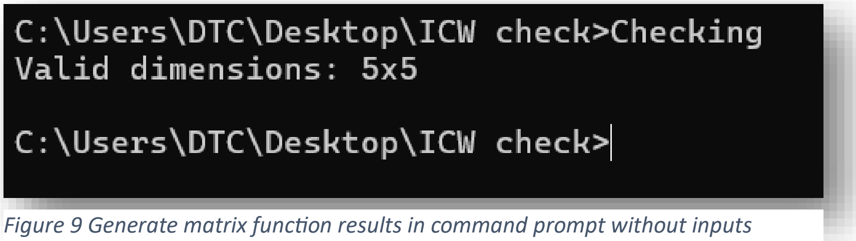
```
                    print("Error: Dimensions must be between '3 3' and
                    return
            except ValueError:
                print("Error: Invalid input format.")
                return

    print(f"Valid dimensions: {num_rows}x{num_cols}")

    # Generate matrix based on user-specified or default dimensions
    matrix = generate_matrix(num_rows, num_cols)
```

Figure 7 Calling generate matrix function in the main function



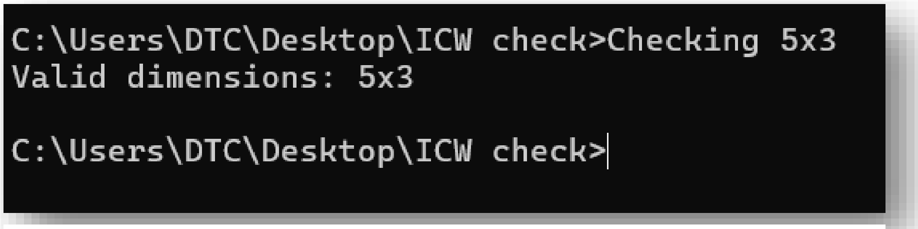
```
============== RESTART: C:\Users\DTC\Desktop\ICW check\Checking.py =======
Valid dimensions: 5x5
>>> 
```

Figure 8 Generate matrix function results in IDLE



```
C:\Users\DTC\Desktop\ICW check>Checking
Valid dimensions: 5x5

C:\Users\DTC\Desktop\ICW check>
```

Figure 9 Generate matrix function results in command prompt without inputs

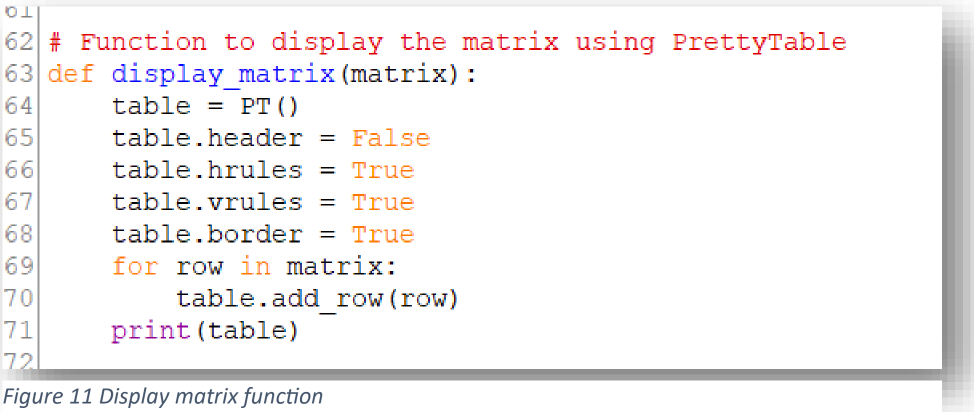


```
C:\Users\DTC\Desktop\ICW check>Checking 5x3
Valid dimensions: 5x3

C:\Users\DTC\Desktop\ICW check>
```

Figure 10 Generate matrix function results in command prompt with inputs

4.4Displaying the Matrix

```
# Function to display the matrix using PrettyTable

def display_matrix(matrix):

    table = PT()

    table.header = False

    table.hrules = True

    table.vrules = True

    table.border = True

    for row in matrix:

        table.add_row(row)

    print(table)
```

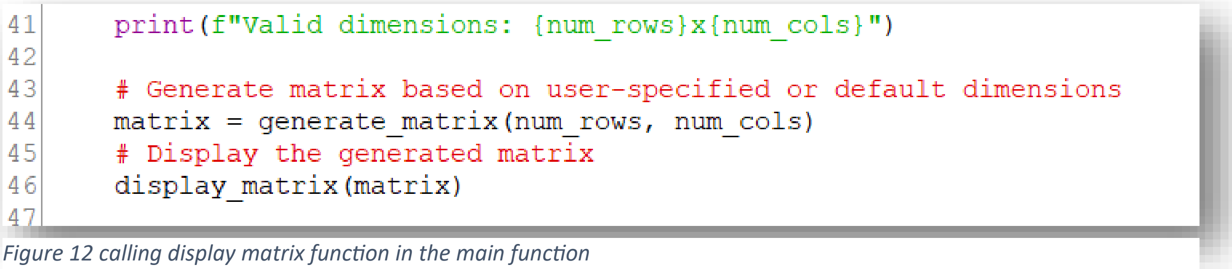


```
# Function to display the matrix using PrettyTable
def display_matrix(matrix):
    table = PT()
    table.header = False
    table.hrules = True
    table.vrules = True
    table.border = True
    for row in matrix:
        table.add_row(row)
    print(table)
```

Figure 11 Display matrix function

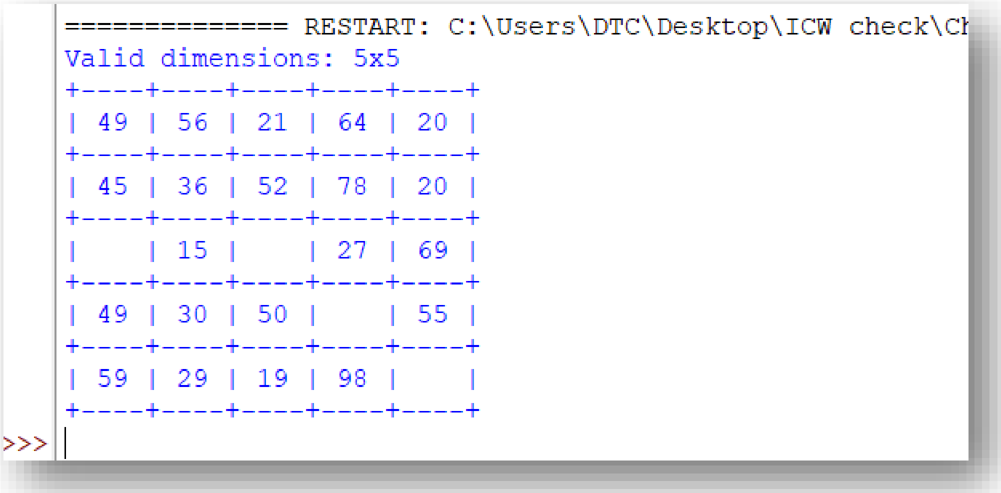
Display the generated matrix

display_matrix(matrix)



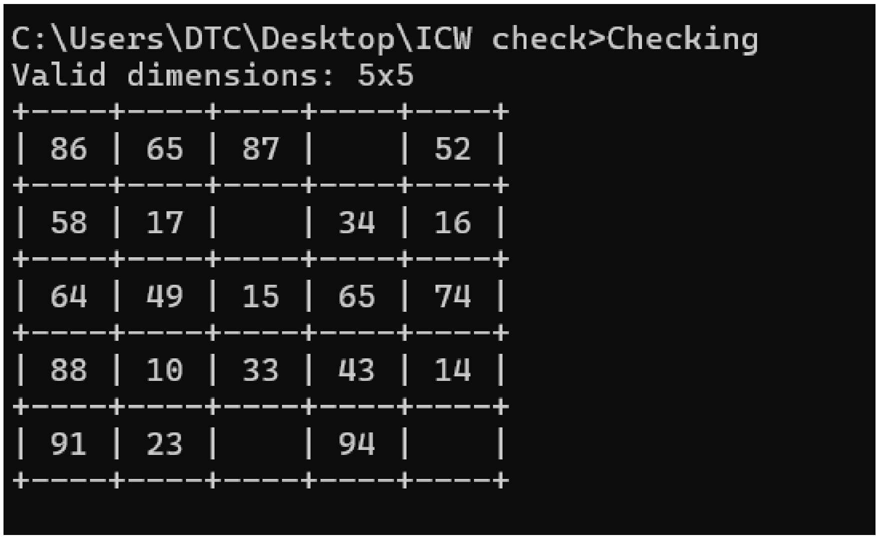
```
    print(f"Valid dimensions: {num_rows}x{num_cols}")

    # Generate matrix based on user-specified or default dimensions
    matrix = generate_matrix(num_rows, num_cols)
    # Display the generated matrix
    display_matrix(matrix)
```

Figure 12 calling display matrix function in the main function



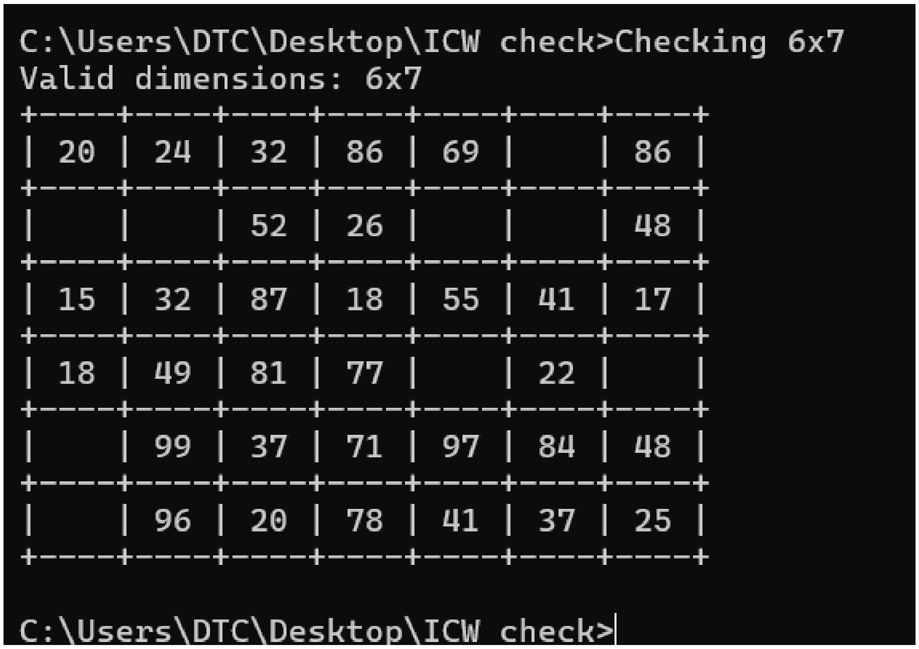
```
============== RESTART: C:\Users\DTC\Desktop\ICW check\Ch
Valid dimensions: 5x5
+----+----+----+----+----+
| 49 | 56 | 21 | 64 | 20 |
+----+----+----+----+----+
| 45 | 36 | 52 | 78 | 20 |
+----+----+----+----+----+
|    | 15 |    | 27 | 69 |
+----+----+----+----+----+
| 49 | 30 | 50 |    | 55 |
+----+----+----+----+----+
| 59 | 29 | 19 | 98 |    |
+----+----+----+----+----+
>>> 
```

Figure 13 Display matrix function results in IDLE



```
C:\Users\DTC\Desktop\ICW check>Checking
Valid dimensions: 5x5
+----+----+----+----+----+
| 86 | 65 | 87 |    | 52 |
+----+----+----+----+----+
| 58 | 17 |    | 34 | 16 |
+----+----+----+----+----+
| 64 | 49 | 15 | 65 | 74 |
+----+----+----+----+----+
| 88 | 10 | 33 | 43 | 14 |
+----+----+----+----+----+
| 91 | 23 |    | 94 |    |
+----+----+----+----+----+
```

Figure 14 Display matrix function results in command prompt without inputs

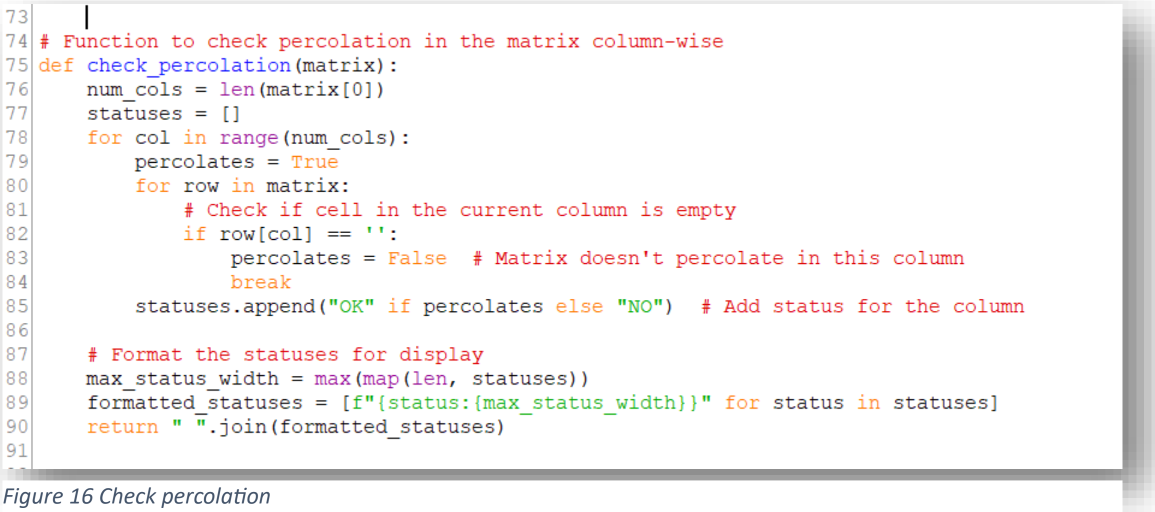


```
C:\Users\DTC\Desktop\ICW check>Checking 6x7
Valid dimensions: 6x7
+----+----+----+----+----+----+----+
| 20 | 24 | 32 | 86 | 69 |    | 86 |
+----+----+----+----+----+----+----+
|    |    | 52 | 26 |    |    | 48 |
+----+----+----+----+----+----+----+
| 15 | 32 | 87 | 18 | 55 | 41 | 17 |
+----+----+----+----+----+----+----+
| 18 | 49 | 81 | 77 |    | 22 |    |
+----+----+----+----+----+----+----+
|    | 99 | 37 | 71 | 97 | 84 | 48 |
+----+----+----+----+----+----+----+
|    | 96 | 20 | 78 | 41 | 37 | 25 |
+----+----+----+----+----+----+----+

C:\Users\DTC\Desktop\ICW check>
```

Figure 15 Display matrix function results in command prompt with inputs

4.5Checking Percolation

```
# Function to check percolation in the matrix column-wise
def check_percolation(matrix):
    num_cols = len(matrix[0])
    statuses = []
    for col in range(num_cols):
        percolates = True
        for row in matrix:
            # Check if cell in the current column is empty
            if row[col] == '':
                percolates = False  # Matrix doesn't percolate in this column
                break
        statuses.append("OK" if percolates else "NO")  # Add status for the column

    # Format the statuses for display
    max_status_width = max(map(len, statuses))
    formatted_statuses = [f"{status:{max_status_width}}" for status in statuses]
    return " ".join(formatted_statuses)
```

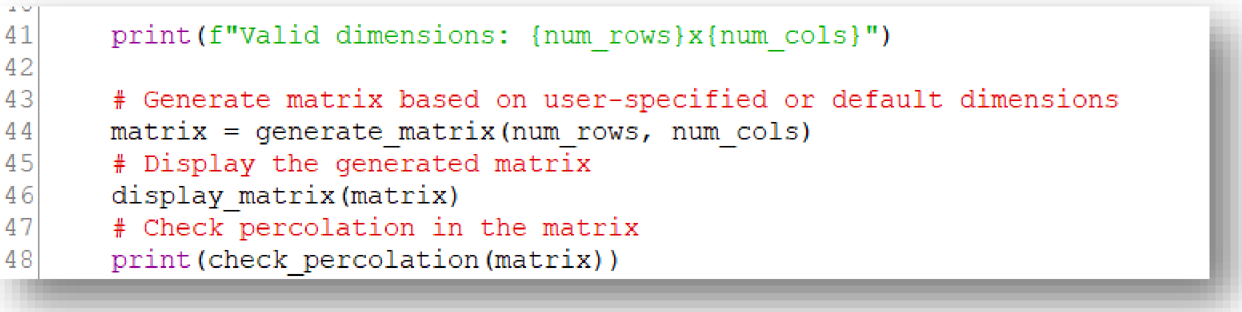


```python

# Function to check percolation in the matrix column-wise
def check_percolation(matrix):
    num_cols = len(matrix[0])
    statuses = []
    for col in range(num_cols):
        percolates = True
        for row in matrix:
            # Check if cell in the current column is empty
            if row[col] == '':
                percolates = False  # Matrix doesn't percolate in this column
                break
        statuses.append("OK" if percolates else "NO")  # Add status for the column

    # Format the statuses for display
    max_status_width = max(map(len, statuses))
    formatted_statuses = [f"{status:{max_status_width}}" for status in statuses]
    return " ".join(formatted_statuses)

```

Figure 16 Check percolation

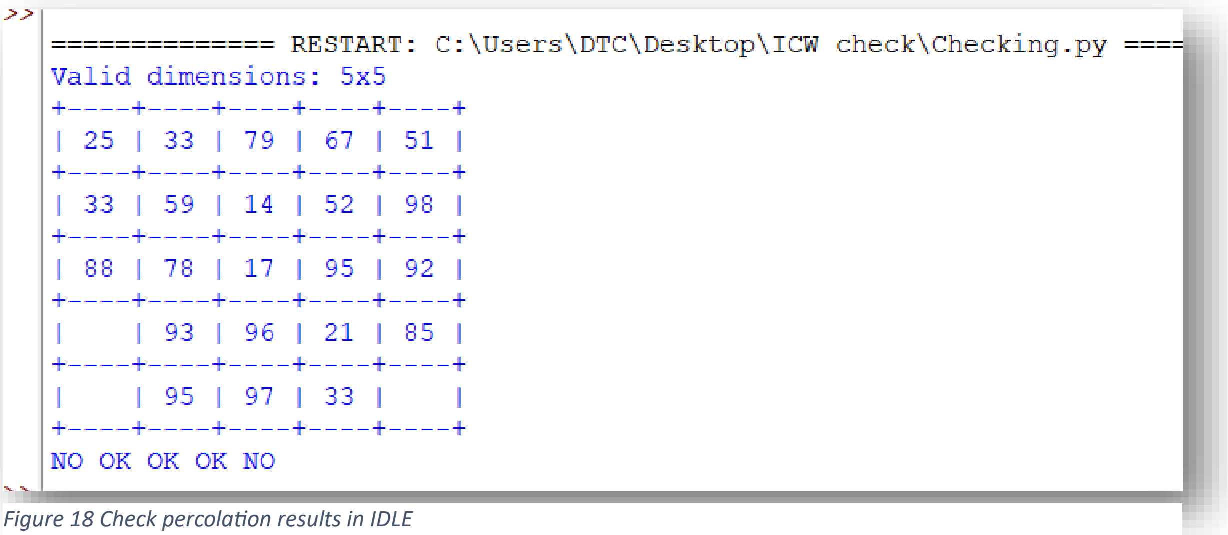
#Check percolation in the matrix

print(check_percolation(matrix))



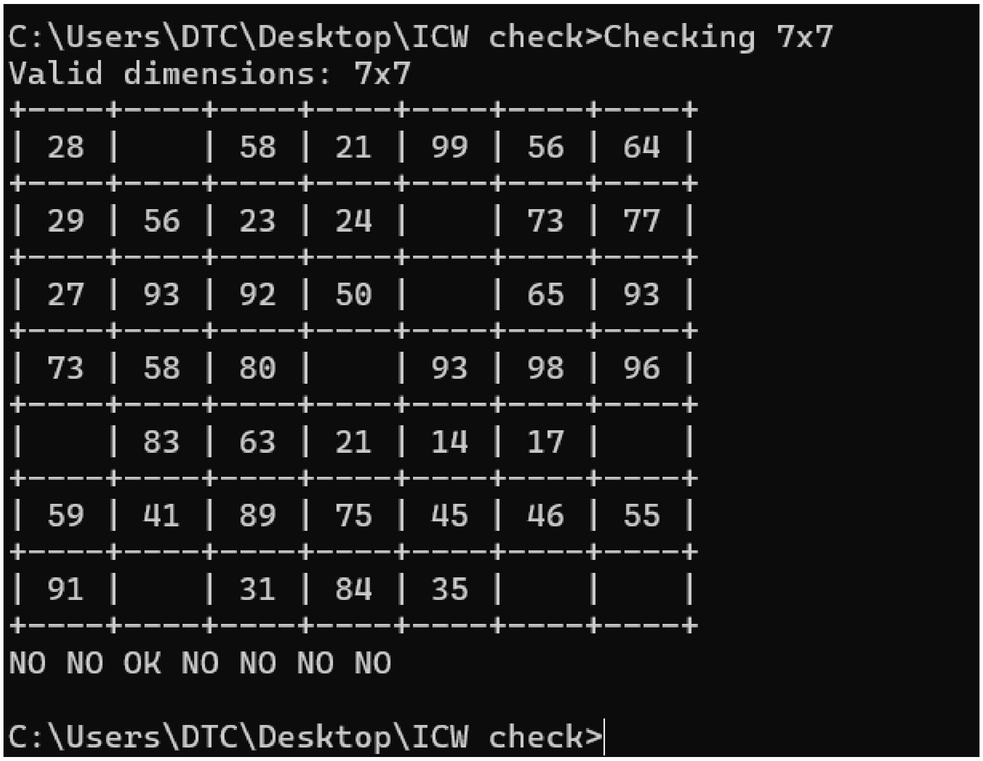
```python
    print(f"Valid dimensions: {num_rows}x{num_cols}")

    # Generate matrix based on user-specified or default dimensions
    matrix = generate_matrix(num_rows, num_cols)
    # Display the generated matrix
    display_matrix(matrix)
    # Check percolation in the matrix
    print(check_percolation(matrix))
```

Figure 17 calling check percolation function in the main function



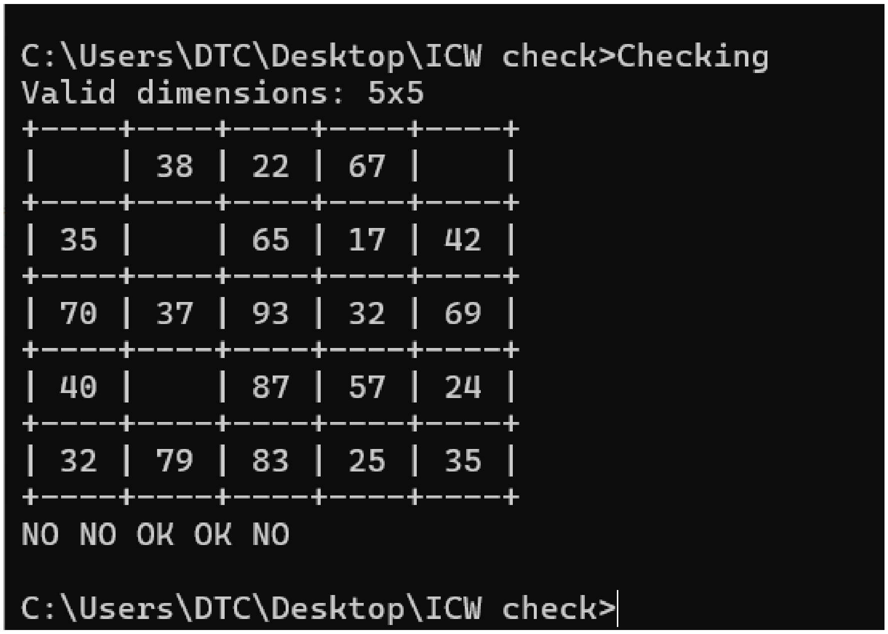
```
>>
============== RESTART: C:\Users\DTC\Desktop\ICW check\Checking.py ====
Valid dimensions: 5x5
+----+----+----+----+----+
| 25 | 33 | 79 | 67 | 51 |
+----+----+----+----+----+
| 33 | 59 | 14 | 52 | 98 |
+----+----+----+----+----+
| 88 | 78 | 17 | 95 | 92 |
+----+----+----+----+----+
|    | 93 | 96 | 21 | 85 |
+----+----+----+----+----+
|    | 95 | 97 | 33 |    |
+----+----+----+----+----+
NO OK OK OK NO
```

Figure 18 Check percolation results in IDLE



```
C:\Users\DTC\Desktop\ICW check>Checking 7x7
Valid dimensions: 7x7
+----+----+----+----+----+----+----+
| 28 |    | 58 | 21 | 99 | 56 | 64 |
+----+----+----+----+----+----+----+
| 29 | 56 | 23 | 24 |    | 73 | 77 |
+----+----+----+----+----+----+----+
| 27 | 93 | 92 | 50 |    | 65 | 93 |
+----+----+----+----+----+----+----+
| 73 | 58 | 80 |    | 93 | 98 | 96 |
+----+----+----+----+----+----+----+
|    | 83 | 63 | 21 | 14 | 17 |    |
+----+----+----+----+----+----+----+
| 59 | 41 | 89 | 75 | 45 | 46 | 55 |
+----+----+----+----+----+----+----+
| 91 |    | 31 | 84 | 35 |    |    |
+----+----+----+----+----+----+----+
NO NO OK NO NO NO NO

C:\Users\DTC\Desktop\ICW check>
```

Figure 19 Check percolation function results in command prompt with inputs

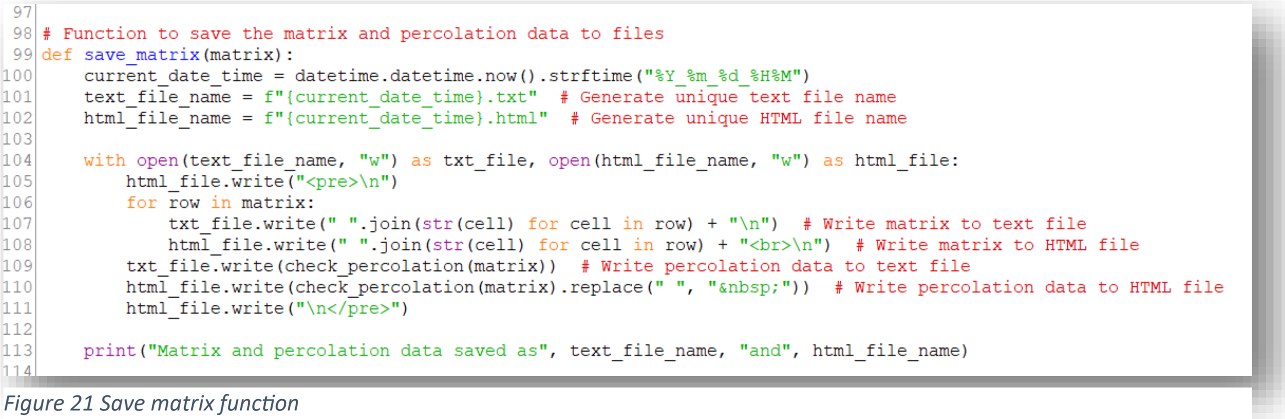


```
C:\Users\DTC\Desktop\ICW check>Checking
Valid dimensions: 5x5
+----+----+----+----+----+
|    | 38 | 22 | 67 |    |
+----+----+----+----+----+
| 35 |    | 65 | 17 | 42 |
+----+----+----+----+----+
| 70 | 37 | 93 | 32 | 69 |
+----+----+----+----+----+
| 40 |    | 87 | 57 | 24 |
+----+----+----+----+----+
| 32 | 79 | 83 | 25 | 35 |
+----+----+----+----+----+
NO NO OK OK NO

C:\Users\DTC\Desktop\ICW check>
```

Figure 20 Check percolation function results in command prompt without inputs

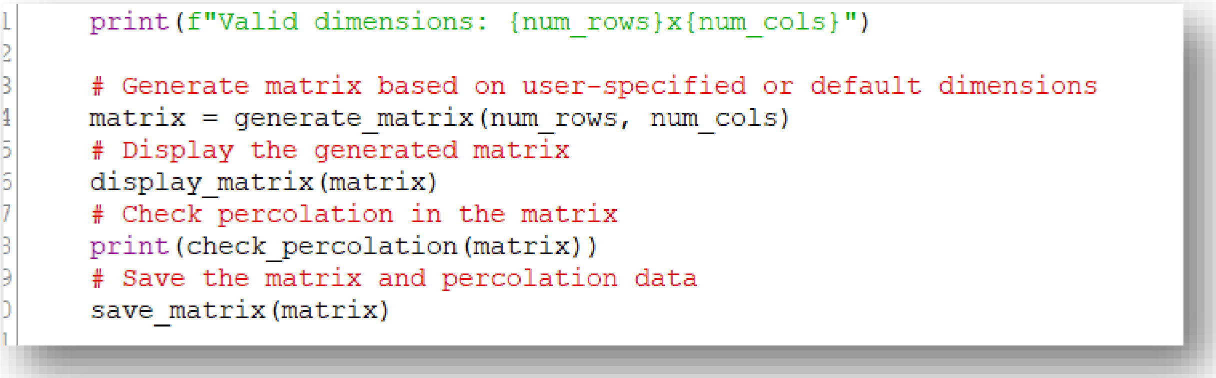
4.6Saving the Matrix

```
# Function to save the matrix and percolation data to files

def save_matrix(matrix):

    current_date_time = datetime.datetime.now().strftime("%Y_%m_%d_%H%M")

    text_file_name = f"{current_date_time}.txt"  # Generate unique text file name

    html_file_name = f"{current_date_time}.html"  # Generate unique HTML file name
    with open(text_file_name, "w") as txt_file, open(html_file_name, "w") as html_file:
        html_file.write("<pre>\n")
        for row in matrix:
            txt_file.write(" ".join(str(cell) for cell in row) + "\n")  # Write matrix to text file
            html_file.write(" ".join(str(cell) for cell in row) + "<br>\n")  # Write matrix to HTML file
        txt_file.write(check_percolation(matrix))  # Write percolation data to text file
        html_file.write(check_percolation(matrix).replace(" ", "&nbsp;"))  # Write percolation data to HTML file
        html_file.write("\n</pre>")

    print("Matrix and percolation data saved as", text_file_name, "and", html_file_name)
```



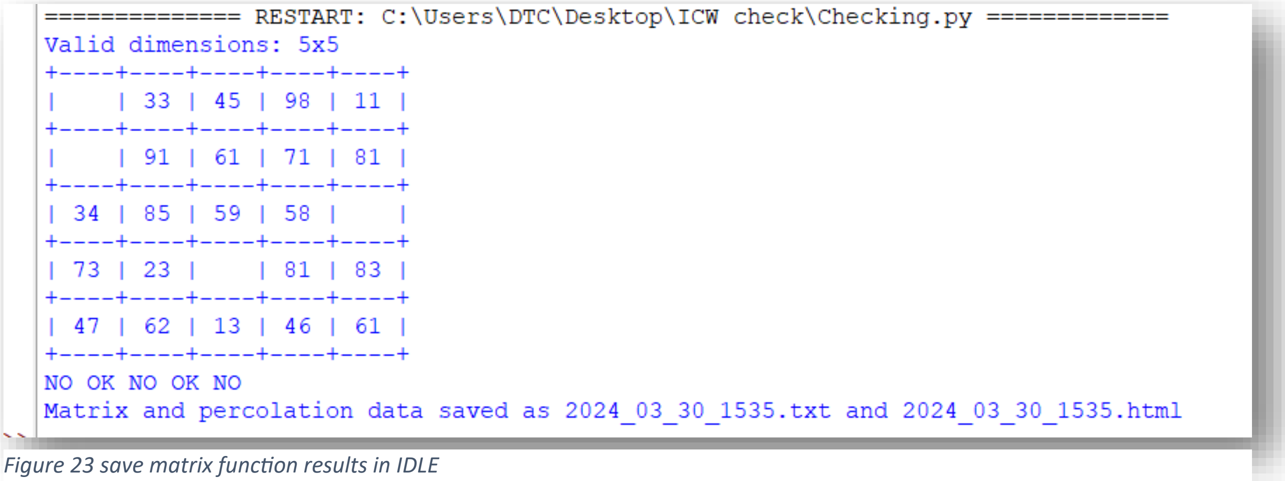
```

# Function to save the matrix and percolation data to files
def save_matrix(matrix):
    current_date_time = datetime.datetime.now().strftime("%Y_%m_%d_%H%M")
    text_file_name = f"{current_date_time}.txt"  # Generate unique text file name
    html_file_name = f"{current_date_time}.html"  # Generate unique HTML file name

    with open(text_file_name, "w") as txt_file, open(html_file_name, "w") as html_file:
        html_file.write("<pre>\n")
        for row in matrix:
            txt_file.write(" ".join(str(cell) for cell in row) + "\n")  # Write matrix to text file
            html_file.write(" ".join(str(cell) for cell in row) + "<br>\n")  # Write matrix to HTML file
        txt_file.write(check_percolation(matrix))  # Write percolation data to text file
        html_file.write(check_percolation(matrix).replace(" ", "&nbsp;"))  # Write percolation data to HTML file
        html_file.write("\n</pre>")

    print("Matrix and percolation data saved as", text_file_name, "and", html_file_name)

```

Figure 21 Save matrix function



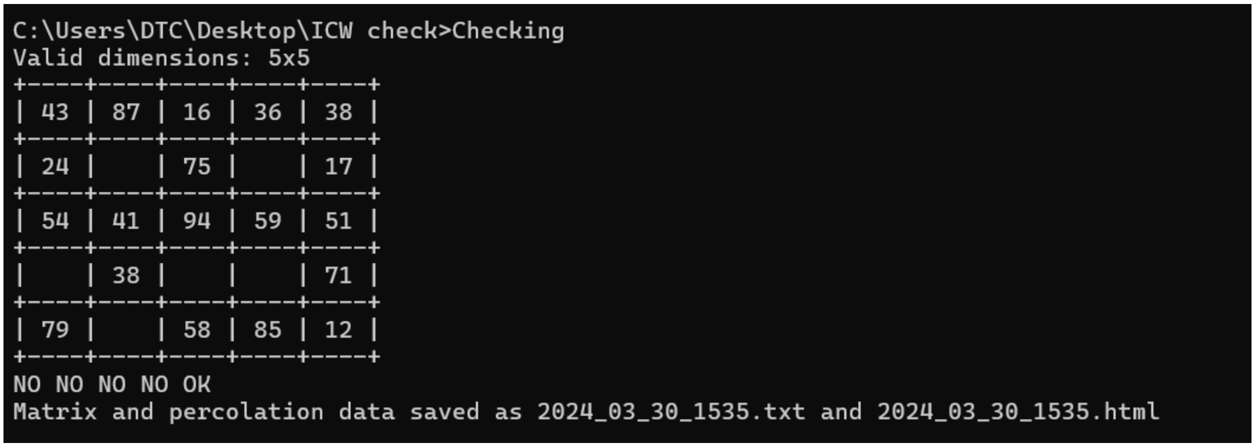
```
    print(f"Valid dimensions: {num_rows}x{num_cols}")

    # Generate matrix based on user-specified or default dimensions
    matrix = generate_matrix(num_rows, num_cols)
    # Display the generated matrix
    display_matrix(matrix)
    # Check percolation in the matrix
    print(check_percolation(matrix))
    # Save the matrix and percolation data
    save_matrix(matrix)
```

Figure 22 calling save matrix function in the main function



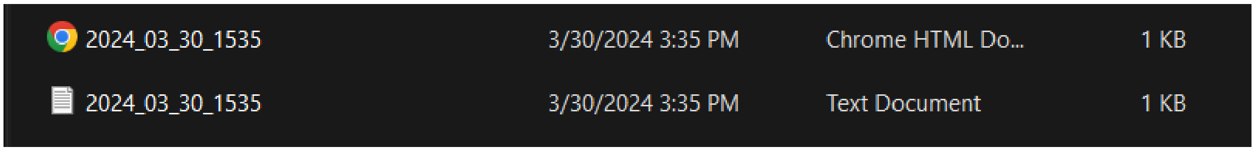
```
============== RESTART: C:\Users\DTC\Desktop\ICW check\Checking.py =============
Valid dimensions: 5x5
+----+----+----+----+----+
|    | 33 | 45 | 98 | 11 |
+----+----+----+----+----+
|    | 91 | 61 | 71 | 81 |
+----+----+----+----+----+
| 34 | 85 | 59 | 58 |    |
+----+----+----+----+----+
| 73 | 23 |    | 81 | 83 |
+----+----+----+----+----+
| 47 | 62 | 13 | 46 | 61 |
+----+----+----+----+----+
NO OK NO OK NO
Matrix and percolation data saved as 2024_03_30_1535.txt and 2024_03_30_1535.html
```

Figure 23 save matrix function results in IDLE



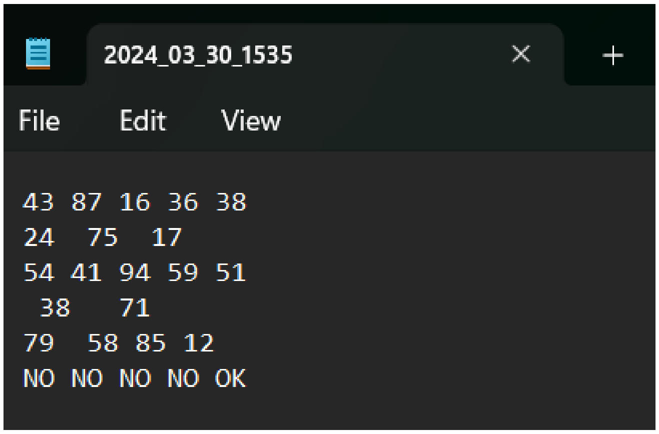
```
C:\Users\DTC\Desktop\ICW check>Checking
Valid dimensions: 5x5
+----+----+----+----+----+
| 43 | 87 | 16 | 36 | 38 |
+----+----+----+----+----+
| 24 |    | 75 |    | 17 |
+----+----+----+----+----+
| 54 | 41 | 94 | 59 | 51 |
+----+----+----+----+----+
|    | 38 |    |    | 71 |
+----+----+----+----+----+
| 79 |    | 58 | 85 | 12 |
+----+----+----+----+----+
NO NO NO NO OK
Matrix and percolation data saved as 2024_03_30_1535.txt and 2024_03_30_1535.html
```

Figure 24 save matrix function results in command prompt without inputs



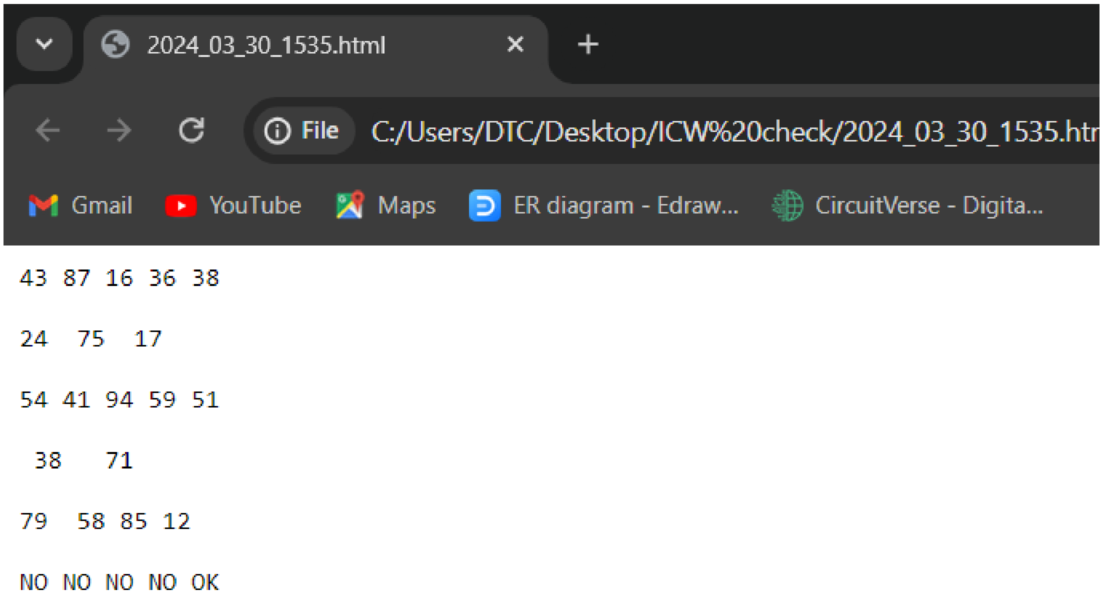
2024_03_30_1535	3/30/2024 3:35 PM	Chrome HTML Do...	1 KB
2024_03_30_1535	3/30/2024 3:35 PM	Text Document	1 KB

Figure 25 Saved as html file and in notepad



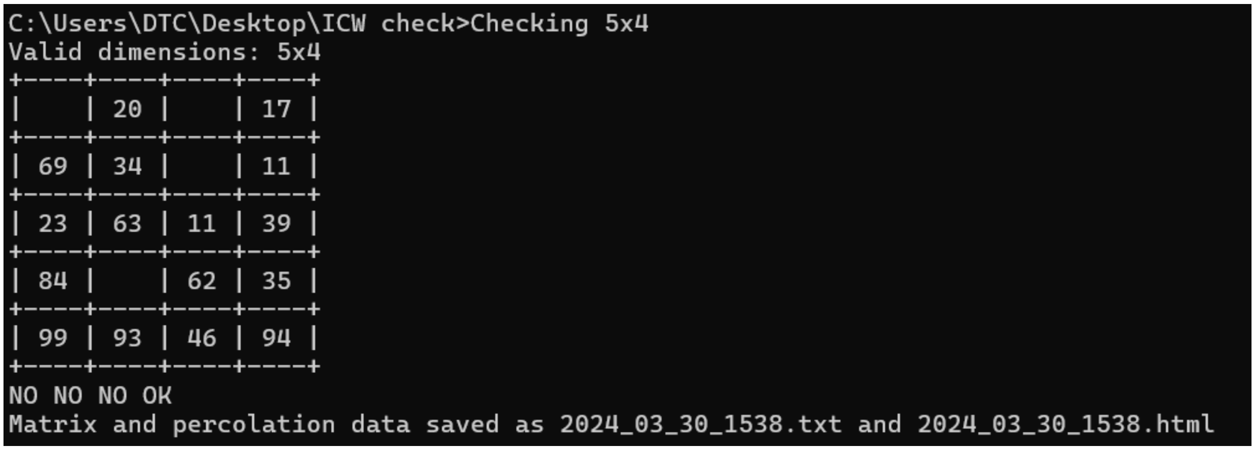
```
43 87 16 36 38
24    75    17
54 41 94 59 51
 38    71
79    58 85 12
NO NO NO NO OK
```

Figure 26 Saved in notepad



```
43 87 16 36 38

24    75    17

54 41 94 59 51

 38    71

79    58 85 12

NO NO NO NO OK
```

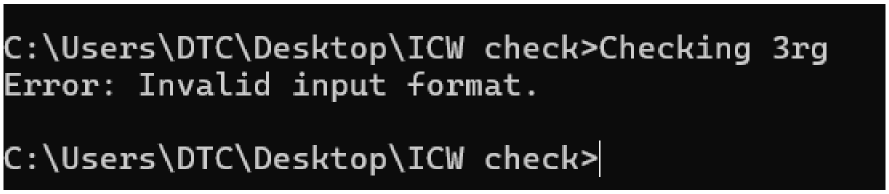
Figure 27 Saved as html file



```
C:\Users\DTC\Desktop\ICW check>Checking 5x4
Valid dimensions: 5x4
+----+----+----+----+
|    | 20 |    | 17 |
+----+----+----+----+
| 69 | 34 |    | 11 |
+----+----+----+----+
| 23 | 63 | 11 | 39 |
+----+----+----+----+
| 84 |    | 62 | 35 |
+----+----+----+----+
| 99 | 93 | 46 | 94 |
+----+----+----+----+
NO NO NO OK
Matrix and percolation data saved as 2024_03_30_1538.txt and 2024_03_30_1538.html
```

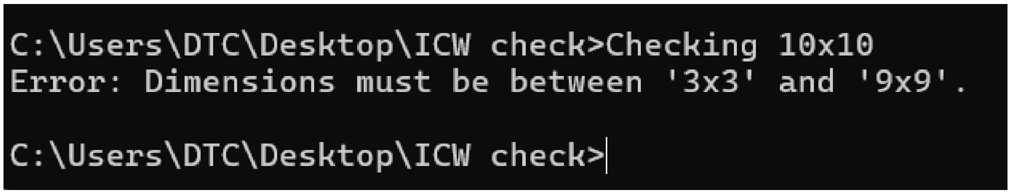
Figure 28 save matrix function results in command prompt with inputs

4.7 Inserting Invalid Inputs



```
C:\Users\DTC\Desktop\ICW check>Checking 3rg
Error: Invalid input format.

C:\Users\DTC\Desktop\ICW check>
```

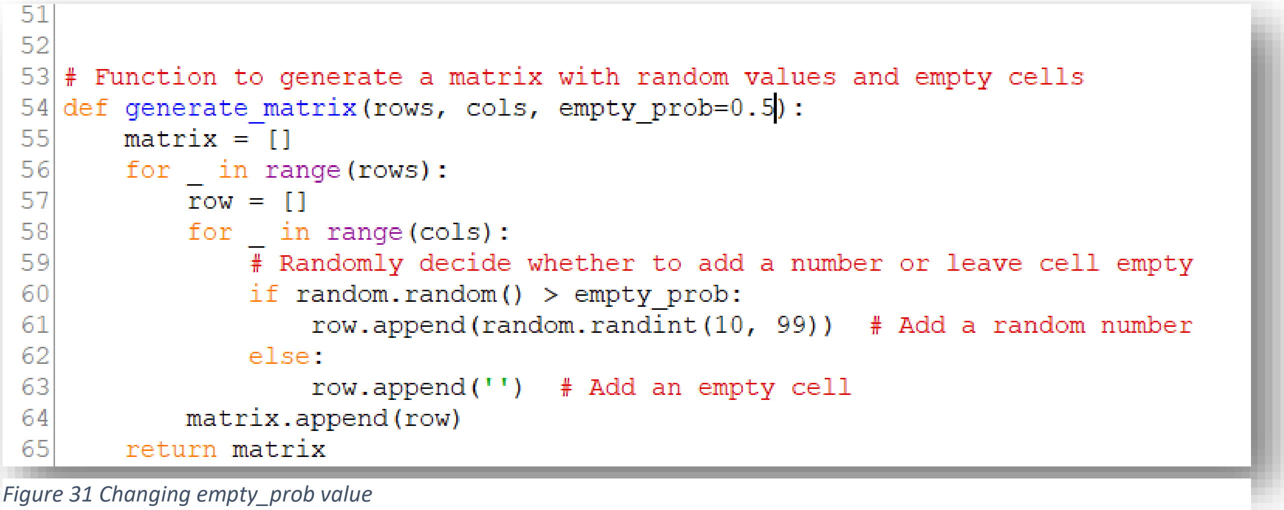
Figure 29 Inserting invalid inputs



```
C:\Users\DTC\Desktop\ICW check>Checking 10x10
Error: Dimensions must be between '3x3' and '9x9'.

C:\Users\DTC\Desktop\ICW check>
```

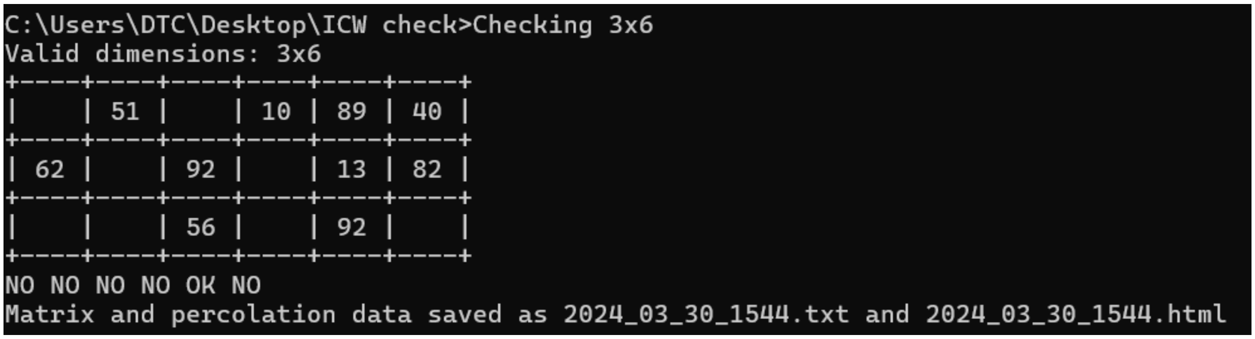
Figure 30 Inserting inputs out of range

4.8 Changing the value of empty_prob



```python


# Function to generate a matrix with random values and empty cells
def generate_matrix(rows, cols, empty_prob=0.5):
    matrix = []
    for _ in range(rows):
        row = []
        for _ in range(cols):
            # Randomly decide whether to add a number or leave cell empty
            if random.random() > empty_prob:
                row.append(random.randint(10, 99))  # Add a random number
            else:
                row.append('')  # Add an empty cell
        matrix.append(row)
    return matrix
```

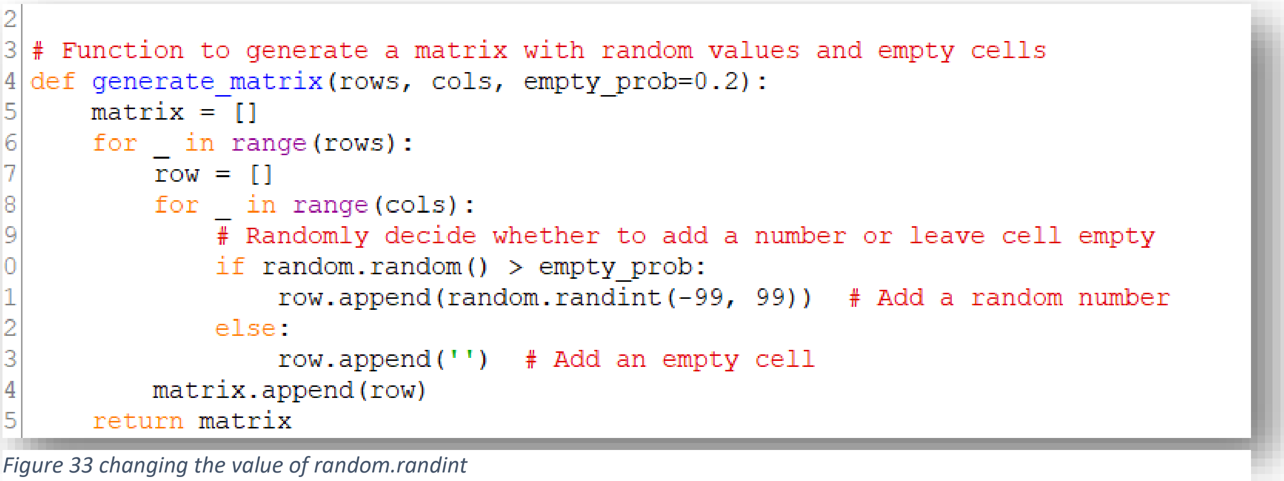
Figure 31 Changing empty_prob value



```
C:\Users\DTC\Desktop\ICW check>Checking 3x6
Valid dimensions: 3x6
+----+----+----+----+----+----+
|    | 51 |    | 10 | 89 | 40 |
+----+----+----+----+----+----+
| 62 |    | 92 |    | 13 | 82 |
+----+----+----+----+----+----+
|    |    | 56 |    | 92 |    |
+----+----+----+----+----+----+
NO NO NO NO OK NO
Matrix and percolation data saved as 2024_03_30_1544.txt and 2024_03_30_1544.html
```

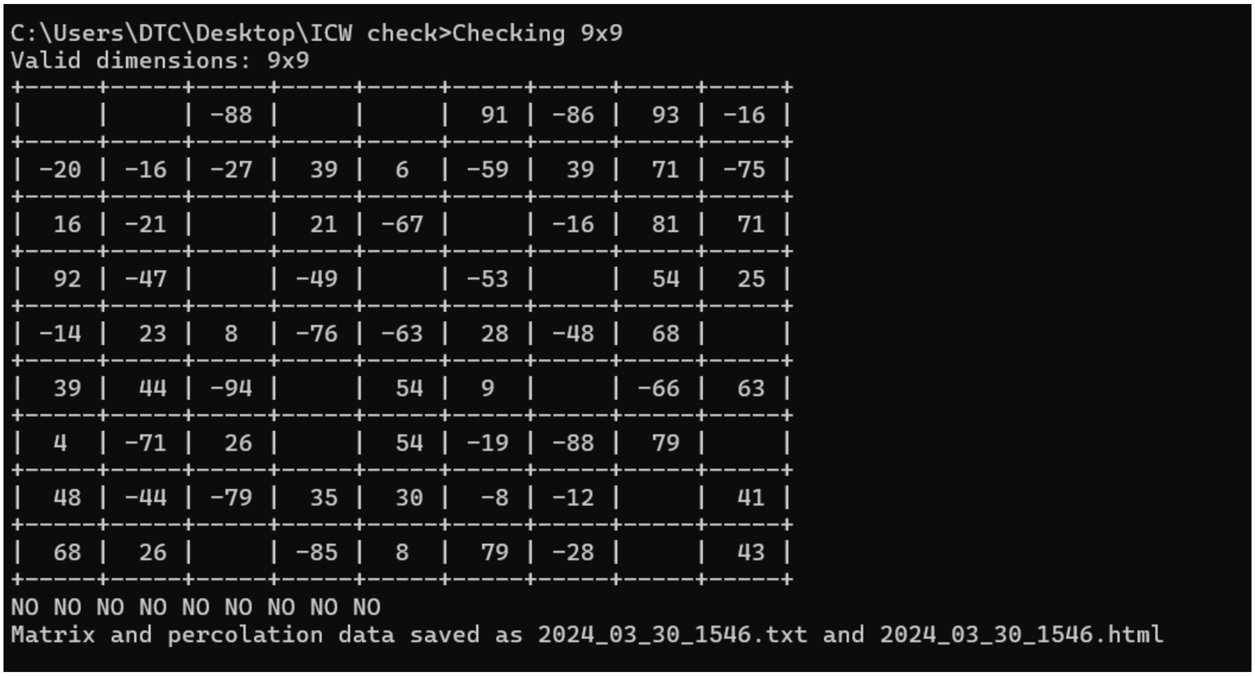
Figure 32 changing empty_prob value results

4.9Changing the value of random.randint



```
# Function to generate a matrix with random values and empty cells
def generate_matrix(rows, cols, empty_prob=0.2):
    matrix = []
    for _ in range(rows):
        row = []
        for _ in range(cols):
            # Randomly decide whether to add a number or leave cell empty
            if random.random() > empty_prob:
                row.append(random.randint(-99, 99))  # Add a random number
            else:
                row.append('')  # Add an empty cell
        matrix.append(row)
    return matrix
```

Figure 33 changing the value of random.randint



```
C:\Users\DTC\Desktop\ICW check>Checking 9x9
Valid dimensions: 9x9
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|     |     | -88 |     |     |  91 | -86 |  93 | -16 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| -20 | -16 | -27 |  39 |  6  | -59 |  39 |  71 | -75 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|  16 | -21 |     |  21 | -67 |     | -16 |  81 |  71 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|  92 | -47 |     | -49 |     | -53 |     |  54 |  25 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| -14 |  23 |  8  | -76 | -63 |  28 | -48 |  68 |     |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|  39 |  44 | -94 |     |  54 |  9  |     | -66 |  63 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|  4  | -71 |  26 |     |  54 | -19 | -88 |  79 |     |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|  48 | -44 | -79 |  35 |  30 |  -8 | -12 |     |  41 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|  68 |  26 |     | -85 |  8  |  79 | -28 |     |  43 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
NO NO NO NO NO NO NO NO NO
Matrix and percolation data saved as 2024_03_30_1546.txt and 2024_03_30_1546.html
```

Figure 34 Changing the value of random.randint results

5. Test Cases

Table 1 Test cases

Test Scenario	**Expected Result**	**Actual Result**	**Pass/Fail**
Input: ‘5x5’	5x5 matrix with random values	5x5 matrix with random values	Pass
Input: ‘3*3’	Invalid input format. Please use the format ‘NxM’	Invalid input format. Please use the format ‘NxM’	Pass
Input: ‘2x4’	Dimensions must be between ‘3x3’ and ‘9x9’	Dimensions must be between ‘3x3’ and ‘9x9’	Pass
Input: 3x3	A randomly generated 3x3 matrix	A randomly generated 3x3 matrix	Pass
Input: 7x7	A randomly generated 7x7 matrix	A randomly generated 7x7 matrix	Pass
A 4x4 matrix with numbers in each column (percolation possible)	“OK OK OK OK” (percolation status for each column)	“OK OK OK OK” (percolation status for each column)	Pass
A 3x3 matrix with empty cells in one column (percolation not possible)	“NO NO NO” (percolation status for each column)	“NO NO NO” (percolation status for each column)	Pass

Test Saving Matrix and Percolation Data to Files	Confirmation message with file names	Confirmation message with file names	Pass
Test File Contents for Correctness	Have the same output of command prompt in the files	Have the same output of command prompt in the files	Pass
Set empty_prob = 0.5 in generate_matrix function to increase empty cell probability	A matrix with more empty cells than numbers	A matrix with more empty cells than numbers	Pass
Set random.randint(-99, 99) in generate_matrix function to allow negative numbers	A matrix with random negative and positive two-digit numbers	A matrix with random negative and positive two-digit numbers and one digit numbers	Fail

5. Conclusion

In conclusion, this Python program successfully demonstrates a simplified percolation process akin to the flow of liquids through a filter, as commonly observed in coffee-making. The program allows users to input dimensions for a dynamic grid, generates a matrix with random two-digit numbers and empty spaces, and checks each column for percolation feasibility.

Through thorough testing and analysis of the code, it was observed that the program effectively handles various scenarios, including valid and invalid input dimensions, matrix generation with different empty space probabilities, percolation checks for both percolatable and non-percolatable matrices, and saving the matrix data along with percolation status to text and HTML files.

The algorithmic approach used in the program leverages fundamental concepts of Python programming, such as input validation, random number generation, matrix manipulation, and file handling. The code structure is organized and modular, facilitating easy understanding, maintenance, and scalability.

Overall, this project has provided valuable insights into programming logic, data manipulation, and simulation of real-world phenomena. It serves as an excellent educational tool for understanding basic concepts in fluid dynamics and algorithm design, making it a worthwhile endeavor in the realm of computer programming education.

6. References

Julles, T. (2023) Python — Converts a text file to HTML format. Available at: https://www.linkedin.com/pulse/python-converts-text-file-html-format-techwith-julles.

Converting text file to html file with python (no date). Available at: https://stackoverflow.com/questions/24715027/converting-text-file-to-html-file-with-python.

How to write to an HTML file in Python? (no date). Available at: https://www.tutorialspoint.com/how-to-write-to-an-html-file-in-python.

Programming with Mosh (2019) Python Tutorial - Python full course for beginners. Available at: https://www.youtube.com/watch?v=_uQrJ0TkZlc.