CSCE 110: Programming I

Lab #10 (100 points)

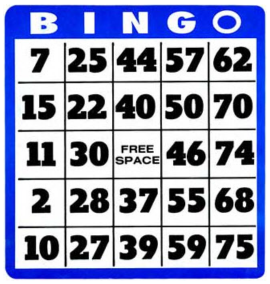
Due: Wednesday, November 29th by 11:59pm

1 Please make sure you understand the following.

Please label your Python programs `q<num>.py`, where `<num>` is the question number. Take your time and make sure you understand everything in this lab before getting started. Also, make sure your programs match the output EXACTLY as given for each question.

2 Lab Questions

1. *Bingo.* Write a program (called `q1.py`) that simulates playing Bingo. In Bingo, each player has a card of numbers arranged in a 5×5 grid with five randomly-chosen numbers from 1 to 15 in the `B` column, 16 to 30 in the `I` column, 31 to 45 in the `N` column, 46 to 60 in the `G` column, and 61 to 75 in the `O` column. The central space is "free" and is considered occupied. The 24 numbers on a Bingo card are unique. An example Bingo card is shown below.



B	I	N	G	O
7	25	44	57	62
15	22	40	50	70
11	30	FREE SPACE	46	74
2	28	37	55	68
10	27	39	59	75

Figure 1: A Bingo card.

A caller randomly calls numbers from 1 to 75 without replacement, each player marking the corresponding number, if it is present on their card, as occupied. The first player to have five occupied numbers in a row horizontally, in a column vertically, or along either of the two major diagonals is the winner.

How to represent a Bingo card in Python. A dictionary will be used to represent a Bingo card. Here, the keys are the letters `B, I, N, G,` and `O` and the value for each key is a list of five integers. Consider the Bingo card in Figure 1. We would represent the card in Python as follows.

```
card = {'B':[7,15,11,2,10], 'I':[25,22,30,28,27] \
        'N':[44,40,0,37,39], 'G':[57,50,46,55,59] \
        'O':[62,70,74,68,75]}
```

Notice that the free space is represented by the integer 0. When simulating Bingo, we will also use the integer 0 to mark that a number has been called. For example, suppose that O-62 has been called. Then, we would mark out the value 62 for the key `'O'` and replace it with a 0.

```
card = {'B':[7,15,11,2,10], 'I':[25,22,30,28,27] \
        'N':[44,40,0,37,39], 'G':[57,50,46,55,59] \
        'O':[0,70,74,68,75]}
```

Detailed instructions. `q1.py` has been provided for you. Please read the comments in the provided code.

Your job is to write the three functions (`check_bingo(), generate_random_card(), simulate_bingo()`) described below. If you want to use additional user-defined functions in your program, please do so.

a) Write the `check_bingo(card)` function. Here, `card` is a Bingo card represented as a dictionary as described on the previous page. Your code will return True if `card` has Bingo. Otherwise, it will return False. There are five Comma Separated Values (CSV) files (e.g., `bingo1.csv` and `bingo2.csv`) to test your code. Example #1 tests your `check_bingo(card)` function.

b) Write the `generate_random_card()` function. You will write code to randomly generate a Bingo card. Your Bingo card must be represented as a dictionary as described earlier (also see comments in code). Your code will return a dictionary that represents a valid Bingo card. Example #2 tests your `generate_random_card()` function.

   When writing your function, you might find the `random.shuffle()` command for lists helpful.

```
import random

a = [10, 20, 30, 40]
random.shuffle(a)     # randomly shuffles the list a in place
print(a)
```

c) Write the `simulate_bingo(n,k)` function. Complete this function to simulate k games (or trials) of Bingo, where n cards (or players) are active in the simulation. Your function will return three values: the *minimum*, *maximum*, and *average* number of calls required to get Bingo among the k games of n cards. For example, if $n = 10$ and $k = 100$, then your simulation would report results for playing Bingo 100 times among 10 players. Remember, for each of the k games, you should generate a new set of Bingo cards.

   To complete the code for this function, make sure to make use of the previous code in parts (a) and (b) that you have written. Your `simulate_bingo(n,k)` function will be used in Example #3.

d) **Note.** The `main(), print_card()`, and `open_file_command()` have been written for you. These functions drive the program, prints a Bingo card stored in a dictionary, and reads a Bingo card from a file, respectively. There is no need to use these functions in the code you are required to write—except possibly for debugging purposes.

Example #1. This example tests your `check_bingo()` function. The user enters the command `open bingo1.csv` (line 1). The file is read and the contents are printed (lines 2–7). These lines use functions (`open_file_command()` and `print_card(card)`)that have been written for you. Line 9 relies on your `check_bingo()` function, which should report that Bingo is found. Next, the file `bingo3.csv` is opened (line 10) and printed (lines 11–16). Bingo is not found for this card. The user quits the program at line 19.

```
 1  > open bingo1.csv
 2   B   I   N   G   O
 3   0  16  34  47  65
 4   0  20  37  50  66
 5   0  25   0  51  67
 6   0  27  40  60  70
 7   0  29  44  61  72
 8
 9  Bingo: True
10  > open bingo3.csv
11   B   I   N   G   O
12   1  19   0  49   0
13   4   0  35  51  69
14   8  26   0   0  71
15  11   0  41  56  73
16  12   0  42  58  74
17
18  Bingo: False
19  > quit
```

Example #2. This example tests your `generate_random_card` function. The user enters `random` (line 1) and a random card is shown (lines 2–7). `random` is entered several more times (lines 8 and 15) and the resulting output is shown (lines 9–14 and 16–21). The user quits the program at line 17.

Note. The cards you randomly generate will be different than those shown here. However, the cards that you generate must be valid Bingo cards.

```
 1  > random
 2   B   I   N   G   O
 3   5  20  31  60  71
 4  12  27  42  46  74
 5  10  25   0  59  61
 6   6  26  36  55  65
 7   7  22  33  47  73
 8  > random
 9   B   I   N   G   O
10   4  18  40  58  68
11   7  28  42  56  69
```

```
12 12  17   0  46  62
13  5  26  38  53  63
14 15  24  31  55  70
15 > random
16  B   I   N   G   O
17 10  30  38  54  70
18  7  18  44  60  63
19 12  27   0  59  73
20  1  29  34  49  67
21 14  21  40  48  62
22 > quit
```

Example #3. This examples tests your `simulate_bingo(n,k)` function. The user types `sim 1 1`, which means simulate 1 player (or 1 card) of 1 game of Bingo. The results of the simulation are shown in lines (2–4). That is, for 1 game of Bingo involving 1 player, 24 calls (or numbers) were required before Bingo was found. Next, the user wants to simulate 1 card involved in 1,000 games of Bingo. The results say that the minimum, maximum, and average number of calls before a player gets Bingo is 11, 67, and 41.4, respectively (lines 6–8). The user studies 50 cards (or players) involved in 100 games of Bingo (lines 9–12) and 250 cards involved in 1,000 games (lines 13–16).

Note: Be prepared to wait if you simulate a lot of cards and Bingo games. Also, your results may differ slightly because of the random nature of simulations.

```
 1 sim 1 1
 2 Minimum: 63.0
 3 Maximum: 63.0
 4 Average: 63.0
 5 > sim 1 100
 6 Minimum: 16.0
 7 Maximum: 59.0
 8 Average: 38.5
 9 > sim 50 100
10 Minimum: 7.0
11 Maximum: 28.0
12 Average: 17.8
13 > sim 250 1000
14 Minimum: 5.0
15 Maximum: 23.0
16 Average: 13.7
17 > quit
```

2. *Sudoku solution checker.* Write a program (called `q2.py`) that determines whether a Sudoku solution is valid or invalid. In Sudoku, the objective is to fill a 9×9 grid with digits so that each row, each column, and each of the nine 3×3 blocks contains all of the digits from 1 to 9 (see Figure 2 below).

5	4	6	8	9	3	2	1	7
3	9	2	1	7	4	6	8	5
8	1	7	5	2	6	9	3	4
7	6	9	3	4	1	8	5	2
2	3	4	7	8	5	1	9	6
1	8	5	2	6	9	7	4	3
4	5	8	6	1	2	3	7	9
9	2	1	4	3	7	5	6	8
6	7	3	9	5	8	4	2	1

Figure 2: Sudoku solution.

The solution will be stored in a file containing 9 lines, where each line represents the solution for a row. For example, consider the file `soln1.txt` below, which contains the solution for the puzzle shown in Figure 2.

soln1.txt

```
546893217
392174685
817526934
769341852
234785196
185269743
458612379
921437568
673958421
```

Since `soln1.txt` represents a valid solution, your program will output "`Valid solution`." If a solution is invalid, your program will output "`Invalid solution`."

Your objective. Sample code has been provided for you in `q1.py`. The provided code (i) asks the user for a file, (ii) stores the data from the file in a nested tuple of tuples, and (iii) prints the results. Your job is to write the remaining pieces of the code to verify the solution by checking the nine rows, nine columns, and nine 3x3 blocks. Thus, the functions you will write are `valid_rows(soln)`, `valid_cols(soln)`, and `valid_blocks(soln)`, respectively.

Several files have been provided to test your program. But, you should also create your own test files to test the correctness of your code. Also, pay attention to the comments in the provided code.

Programming tips. For each function, print the contents of the variable soln (which is a nested tuple of tuples) to see the data you are manipulating. Next, draw a picture of the soln data structure before writing your code. Once you understand how the data is organized, you are ready to write the code.

Example #1. The user enters the name of the Sudoku solution file (line 1). The program then determines that the soln1.txt file is a valid Sudoku solution (line 3).

```
Enter filename: soln1.txt

Valid solution
```

Example #2. The user enters the name of the Sudoku solution file (line 1). The program then determines that the soln3.txt file is an invalid Sudoku solution (line 3).

```
Enter filename: soln3.txt

Invalid solution
```

3. Image Zoom: Write a program (q3.py) that reads the file felix.txt (showed in class) and produces a png image from the file with a zoom specified by the user. The original file has 35 x 35 characters, so it produces a 35 x 35 pixels image. Your program will ask for the size of the zoom and your program should create the image with this zoom. If the user enters 2, your image will have 70 x 70 pixels. Some zoom images are presented as follows:

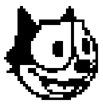


Figure 3: Zoom 1

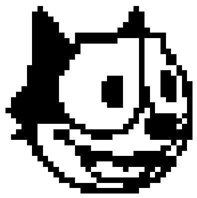


Figure 4: Zoom 2

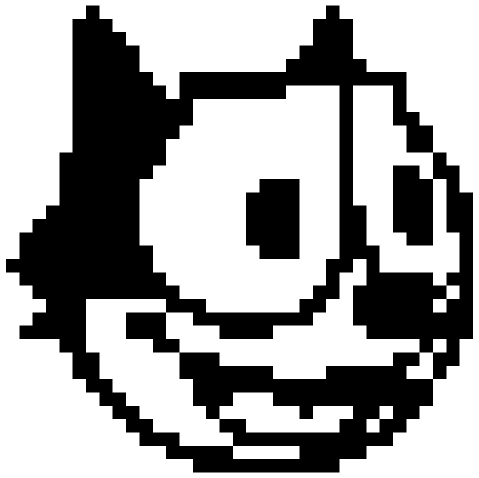


Figure 5: Zoom 5

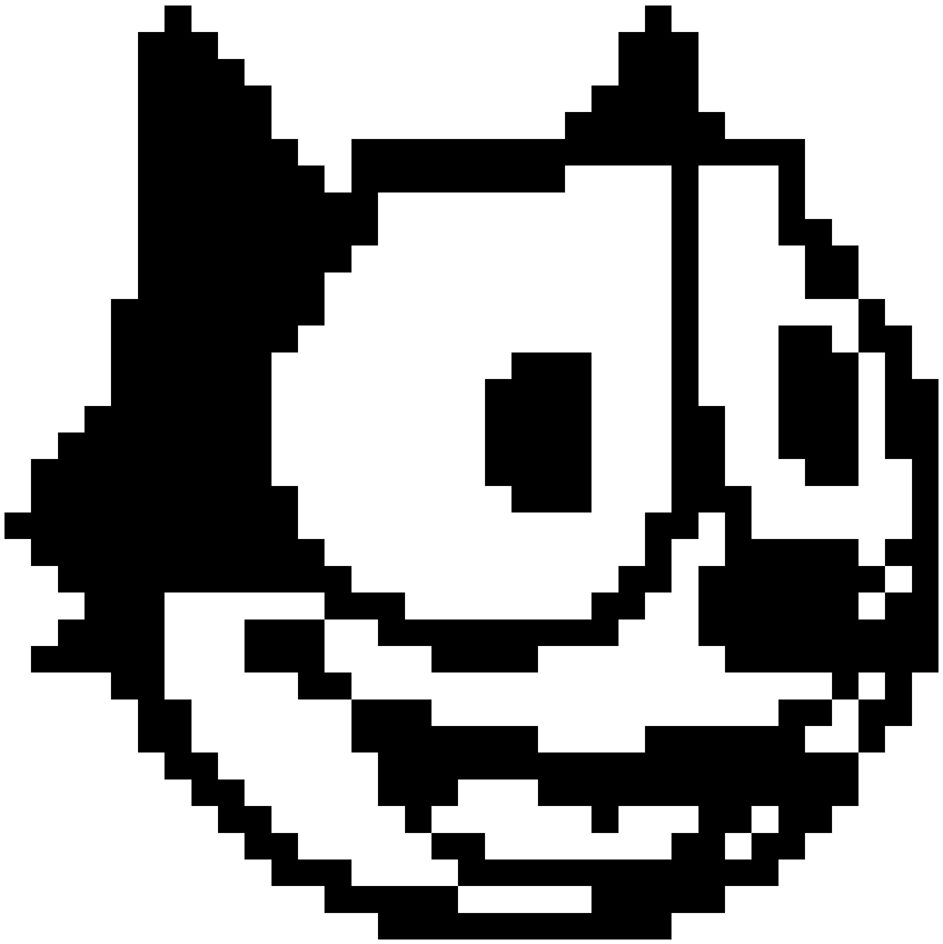


Figure 6: Zoom 10