CSE 1321L: Programming and Problem Solving I Lab

Assignment 3

Module 2 Part 2

Spring 2026

What students will learn

- Problem Solving.
- Basic Program Structure.
- Input and Output with the user.
- Write code that includes while/for loop and nested loops logic.
- Structure program to include conditional logic.

Content

Overview:

For this assignment, you're going to practice making logic in your code. It will include loops and nested loops. In practical terms, this means you're going to expand on the concepts from previous assignments, but also include things like while loop, for loop statements. Again, start early, practice, and ask a lot of questions.

Final note: *Do not cheat*

If your temptation is to look online, don't. Come see us instead and ask questions – we are here to help. Remember, you are going to have to write codes in your future job interviews, so learn it now to secure a high-paying job later.

Assignment3A: Number Hourglass Pattern

Write a Python program that takes a positive number as input and prints number hourglass pattern using nested loops.
The hourglass will be composed of single-digit numbers (0–9) that repeat using the Modulus (%) operator. The overall size of the hourglass is determined by input value entered by the user.

Requirements:

- Your solution must use **nested for loops** exclusively for pattern generation.
- Assume the user will input any positive integer.
- The input value determines the height and width of the hourglass.
- For this shape to work correctly, only odd values are allowed:
  - If the user enters an even number, the program must automatically increase it by 1.
- The hourglass must be center aligned using spaces.
- The hourglass must be filled with increasing single-digit numbers from 0 to 9:
  - Once the number reaches 9, it must restart from 0.
- You must follow exactly the output format shown in the sample output.

Pattern Description

- The top half of the hourglass starts with the widest row.
- Each subsequent row:
  - Increases the number of leading spaces
  - Decreases the number of printed digits
- The middle row contains exactly **one digit**.
- After the middle row:
  - The pattern expands symmetrically
  - Leading spaces decrease
  - Digits increase per row

Hints:

- You will likely need to use the **Modulus** operator multiple times.
- Count:
  - How many **spaces** appear before the numbers on each row
  - How many **digits** are printed on each row

Example runs are shown below. The user input is shown in **red and bold.**

Sample Output 1:

```
Enter an odd number for the size of the hourglass: 5
01234
 567
  8
 901
23456
```

Sample Output 2:

```
Enter an odd number for the size of the hourglass: 15
012345678901234
 5678901234567
  89012345678
   901234567
    8901234
     56789
      012
       3
      456
     78901
    2345678
   901234567
  89012345678
 9012345678901
234567890123456
```

Sample Output 3:

```
Enter an odd number for the size of the hourglass: 6
0123456
 78901
  234
   5
  678
 90123
4567890
```

Assignment3B: BINGO

Write a Python program that reads a 3×3 Bingo card from the user and validates it using nested loops and conditional statements.
The user will enter the Bingo card as three lines, where each line contains 3 integers separated by commas (example: 5,20,31). Your program must display the card and then print whether it is VALID or INVALID.

For this task:

- Your solution must use nested loops for:
  - Reading and dividing each input row into values
    - You can iterate string char by char. More details in Assignment 3 C below.
    - You can append a comma at the end of the input of each line/row
    - You will need to save this input into 9 different variables.
  - Displaying the 3×3 card
  - Validating the rules below
- Use conditional statements to apply validation rules.
- Assume the user will enter integers and only 3 for each row/line. You don't need to validate this.
- Do **not** use lists, arrays, sets, dictionaries, or any collection types.

Rules:

- Rule 1
  - The center position (row 2, column 2) must be 0.
- Rule 2
  - Each column must stay within its allowed range:
    - Column 1 (B): 1–15
    - Column 2 (I): 16–30
    - Column 3 (N): 31–45
- Rule 3
  - In each row, the non-zero values must be **different**.
    - 5,20,5 is invalid (duplicate in the same row).
  - You do not need to check duplicates across different rows.

Output:

- First, print the card in a 3×3 grid format with column labels i.e. BIN
- Then print:
  - VALID BINGO CARD if all rules pass
  - INVALID BINGO CARD if any rule is violated
- If invalid
  - print a short reason message (violated rule).
  - If the BINGO is invalid that means at least one rule is not satisfied.
  - For full credit you need to print all violations.

Hint:

- You can draw a 3*3 grid on paper and understand rules before writing the code.
  - You can have meaningful variables names according to grid cells

A11	A12	A13
A21	A22	A23
A31	A32	A33

- Use an outer loop to process rows.
- Use an inner loop to extract values from comma-separated input.
- Use conditional statements to determine which range applies based on the column.
- For row-duplicate checking, compare the 3 values within the same row using conditions.

Example runs are shown below. The user input is shown in **red and bold.**

Sample Output #1:

```
Enter Bingo row 1 (3 values, comma-separated): 5,20,31
Enter Bingo row 2 (3 values, comma-separated): 12,0,40
Enter Bingo row 3 (3 values, comma-separated): 15,30,45

B   I   N
5   20   31
12   0   40
15   30   45

VALID BINGO CARD
```

Sample Output 2:

```
Enter Bingo row 1 (3 values, comma-separated): 18,20,31
Enter Bingo row 2 (3 values, comma-separated): 12,7,40
Enter Bingo row 3 (3 values, comma-separated): 15,30,30

B   I   N
18   20   31
12   7   40
15   30   30

INVALID BINGO CARD
Reasons:
- FREE space error: center (row 2, col 2) must be 0
- Range error at row 1, column B: 18 (must be 1–15)
- Range error at row 3, column N: 30 (must be 31–45)
- Duplicate in row 3
```

Assignment3C: Letter Frequency Quiz

Write a Python program that plays an **interactive "Letter Frequency Quiz"** with the user.
The program will first prompt the user to input a **sentence** (lowercase letters only, may include spaces).

1. The program will **hide the sentence** but will tell the user how many times a specific letter appears in the sentence.
2. The user must **guess the frequency** of that letter.
3. After each guess, the program will tell the user if their answer is **too high**, **too low**, or **correct**.
4. The quiz continues for **all unique letters in the sentence**.
5. The program ends after the user guesses the frequency of all letters correctly.
6. You **cannot** use lists or arrays, but you can use the len() function. You should only use strings and simple loops. The program must be able to interact with the user by taking input and displaying results after each guess.

For this task:

1. **Initial Input:**
   - Ask the user for a sentence. Ignore spaces for counting.
2. **Iterate over the string:**
   - Count letters - do **not use lists, dictionaries, or sets**.
3. **String Indexing / Loops:**
   - Use loops and string indexing to check occurrences.
   - More details in next section.
4. **Interactive Quiz:**
   - For each unique letter, repeatedly ask the user for the frequency until they get it correct.
5. **Output:**
   - Display feedback after each guess.
     - "Too high" if the guess is higher than the actual frequency
     - "Too low" if the guess is lower
     - "Correct!" when the guess matches the frequency
6. **Game End:**
   - Program will end once all letters are processed.

String Indexing:

In Python, you can access individual characters of a string using indexing. Each character in a string has a unique position, starting from 0.

- For example, in the word "apple", the index of the letter "a" is 0, the letter "p" is 1, and so on.
- You can use string[i] to access the character at position i in a string.
- For example, if you want to get the first letter of the word "apple", you would use word[0], which will give "a".
- Similarly, word[1] will give "p", word[2] will give "p", and so on.

Example runs are shown below. The user input is shown in **red and bold.**

Sample Output #1:

```
[Welcome to the Letter Frequency Quiz]
Enter a sentence (lowercase letters only): hello world
Guess the frequency of letter 'h': 3
Too high!
Guess the frequency of letter 'h': 1
Correct!
Guess the frequency of letter 'e': 1
Correct!
Guess the frequency of letter 'l': 1
Too low!
Guess the frequency of letter 'l': 3
Correct!
Guess the frequency of letter 'o': 2
Correct!
Guess the frequency of letter 'w': 1
Correct!
Guess the frequency of letter 'r': 1
Correct!
Guess the frequency of letter 'd': 1
Correct!
Congratulations! You completed the quiz.
```

Sample Output 2:

```
[Welcome to the Letter Frequency Quiz]
Enter a sentence (lowercase letters only): apple
Guess the frequency of letter 'a': 5
Too high!
Guess the frequency of letter 'a': 1
Correct!
Guess the frequency of letter 'p': 1
Too low!
Guess the frequency of letter 'p': 2
Correct!
Guess the frequency of letter 'l': 1
Correct!
Guess the frequency of letter 'e': 3
Too high!
Guess the frequency of letter 'e': 1
Correct!
Congratulations! You completed the quiz.
```

Submission Instructions:

- Programs must follow the output format provided. This includes each blank line, colons (:), and other symbols.
- Programs must be working correctly.
- Programs must be written in Python.
- Programs must be submitted with the correct **.py** format.
- Programs must be saved in files with the correct file name:
  - Assignment3A.py
  - Assignment3B.py
  - Assignment3C.py
- Programs (source code files) must be uploaded to Gradescope by the due date.