CSI 102: Lab 06

Functions

Anthony Cavallo

10.22.2025

I certify that this lab report is entirely my own work.

Introduction

In Lab 06 of CSI 102, this covers the usage of functions to simplify code and save it so when the time comes, the function can be called with a simple line instead of copying and pasting code. This allows for optimization of games in a rather rudimentary way, but one that is extremely useful universally.

Methods

Task 1: Player Movement

This program lets the user move a player around on a 5x5 grid using simple direction commands. The player starts at position (0, 0) and can move up, down, left, or right until the user decides to stop.

Step 1: Define the function

A function called move_player is defined to control the player's movement. Inside it, two variables (x and y) represent the player's position on the grid, starting at (0, 0).

Step 2: Display instructions

The program explains the controls:

- Type up, down, left, or right to move.
- Type s to stop the game.
  It also prints the starting position.

Step 3: Get the user's direction

The program uses a while loop to keep asking the user for a direction until they type 's'.

Step 4: Move the player

Each direction changes the coordinates:

- up increases y by 1
- down decreases y by 1

- left decreases x by 1
- right increases x by 1
  After each move, the new position is shown.

Step 5: Prevent going out of bounds
The grid is limited to 5 rows and 5 columns (from 0 to 4). If the player tries to move outside these limits, the program prints a warning like “You hit the top boundary!” and doesn’t move the player.

Step 6: Stop and show final position
When the user presses 's', the loop ends, and the program prints the player’s final coordinates.

```python
# Author: Anthony Cavallo
# Date: 10/22/2025
# Description: Player movement in a game.
# Code of honesty: I certify that this lab is entirely my own work.

def move_player():
    x = 0
    y = 0
    grid_size = 5

    print("Player Movement Started! (Grid size: 5x5)")
    print("Controls: up, down, left, right | Press 's' to stop.")
    print("Starting position:", "(", x, ",", y, ")")

    while True:
        direction = input("Enter direction: ")
        direction = direction.lower()

        if direction == 's':
            break
        if direction == 'up':
            if y < grid_size - 1:
                y = y + 1
            else:
                print("You hit the top boundary!")
        elif direction == 'down':
            if y > 0:
                y = y - 1
            else:
                print("You hit the bottom boundary!")
        elif direction == 'left':
            if x > 0:
                x = x - 1
            else:
                print("You hit the left boundary!")
        elif direction == 'right':
            if x < grid_size - 1:
                x = x + 1
            else:
                print("You hit the right boundary!")
        else:
            print("Invalid input! Please enter up, down, left, right, or 's'.")

        print("Current position:", "(", x, ",", y, ")")

    print("Final position:", "(", x, ",", y, ")")
move_player()
```

Task 2: Score Calculator

This program calculates a player's score based on hits, misses, and the difficulty level of the game. It repeats the process for multiple rounds until the user decides to stop.

Step 1: Define the function
A function called calculate_score is defined. It takes three inputs:

- hits: number of successful hits
- misses: number of failed hits
- difficulty_level: a number (1, 2, or 3)

Step 2: Apply the score formula
Depending on the difficulty level, the function calculates the score using one of these formulas:

- Difficulty 1 → hits * 10 - misses * 5
- Difficulty 2 → hits * 15 - misses * 7
- Difficulty 3 → hits * 20 - misses * 10
  If the user enters an invalid difficulty, the function prints an error and skips the calculation.

Step 3: Ask the user for input
The main part of the program uses a while loop to keep running new score calculations. It asks the user for:

- Number of hits
- Number of misses
- Difficulty level (1–3)
  Typing 's' at any time ends the program.

Step 4: Convert inputs and check for errors
The program checks that the user entered numbers and converts them from text to integers before using them in calculations.

Step 5: Show the score
The calculate_score function is called, and the result is printed for the user.

Step 6: Repeat or stop

After showing the score, the loop repeats for another round until the user presses 's', then it prints a message saying the program has ended.

```
# Author: Anthony Cavallo
# Date: 10/22/2025
# Description: Calculation of scores dependent on difficulty.
# Code of honesty: I certify that this lab is entirely my own work.

def calculate_score(hits, misses, difficulty_level):
    if difficulty_level == 1:
        score = hits * 10 - misses * 5
    elif difficulty_level == 2:
        score = hits * 15 - misses * 7
    elif difficulty_level == 3:
        score = hits * 20 - misses * 10
    else:
        print("Invalid difficulty level! Must be 1, 2, or 3.")
        score = None
    return score
def main():
    print("Score Calculation System")
    print("Press 's' at any time to stop.")

    while True:
        hits_input = input("Enter number of hits: ")
        if hits_input.lower() == 's':
            break

        misses_input = input("Enter number of misses: ")
        if misses_input.lower() == 's':
            break

        difficulty_input = input("Enter difficulty level (1-3): ")
        if difficulty_input.lower() == 's':
            break
        if hits_input.isdigit() and misses_input.isdigit() and difficulty_input.isdigit():
            hits = int(hits_input)
            misses = int(misses_input)
            difficulty = int(difficulty_input)
        else:
            print("Please enter valid numbers for hits, misses, and difficulty.")
            continue

        score = calculate_score(hits, misses, difficulty)
        if score is not None:
            print("Your score for this round:", score)
            print("")
    print("Score calculation ended.")
main()
```

Results

Player Movement:

```
sabre@fedora:~/Documents/CSI 102$ /bin/python "/home/sabre/Documents/CSI 102/Lab06/playerMovement.py"
Player Movement Started! (Grid size: 5x5)
Controls: up, down, left, right | Press 's' to stop.
Starting position: ( 0 , 0 )
Enter direction: up
Current position: ( 0 , 1 )
Enter direction: left
You hit the left boundary!
Current position: ( 0 , 1 )
Enter direction:
```

Score Calculator:

```
sabre@fedora:~/Documents/CSI 102$ /bin/python "/home/sabre/Documents/CSI 102/Lab06/scoreCalculation.py"
Score Calculation System
Press 's' at any time to stop.
Enter number of hits: 45
Enter number of misses: 2
Enter difficulty level (1-3): 2
Your score for this round: 661

Enter number of hits:
```

Discussion

This lab helped me review how loops, conditions, and functions work together in Python. It also gave me a better understanding of how to use user input to control a program in real time. The player movement task showed how logic can be applied to control position within limits, while the score calculation task showed how to perform different operations based on user choices. Both activities connected what I've learned in class to how programs can respond to player actions, similar to what happens in simple games.

Challenges

One challenge I faced was keeping track of how the position changed when moving up, down, left, or right. I had to make sure the coordinates didn't go outside the grid, which took a bit of testing. Another challenge was writing the loop for the score system so it would stop only when the user pressed 's'. Sometimes the inputs didn't behave as expected, so I had to check that the data was valid before doing the math. Once I understood that, it became easier to fix the logic.

Conclusion

This lab helped me strengthen my understanding of loops, conditions, and functions in Python. It showed how simple concepts like movement and scoring can be combined to make interactive programs. Overall, it was a good way to apply what I've learned so far in a more practical and hands-on way.

Appendix

Files included: playerMovement.py, scoreCalculation.py