System Requirements Specification Index

For

Minecraft Score Converter

Version 1.0

IIHT Pvt. Ltd.

fullstack@iiht.com

Table of Contents

Minecraft Score Converter Console

System Requirements Specification

1 Project Abstract

Mojang, the creator of Minecraft, requires a scoring system that can handle different types of score formats from various game activities. The Minecraft Score Converter is a Python console application developed to standardize different scoring formats into a unified system. This system converts mining points (stored as text from legacy systems), combat accuracy (stored as decimal numbers), and achievement bonuses (stored in hexadecimal) into a standardized integer format. The application ensures consistent score calculations across different game modes and allows for fair player rankings on the global leaderboard. This standardization tool is crucial for maintaining fair play and accurate player statistics across the Minecraft gaming ecosystem.

2 Business Requirements:

Screen Name	Console input screen
Problem Statement	1. User needs to enter different types of game scores into the application 2. The application should convert all scores to standard integer format 3. The console should handle the following inputs: Mining points (as text), Combat Accuracy (as decimal), Achievement Bonus (as hexadecimal) 4. Format output clearly with proper units

3 Constraints

3.1 Input Requirements

1. Mining Score:
    - Must be stored as string in variable **mining_score**
    - Must contain only digits
    - Example: "100"
2. Combat Score:
    - Must be stored as float in variable **combat_score**
    - Must be non-negative
    - Example: 98.7
3. Achievement Bonus:
    - Must be positive numbers
    - Must be stored as string in **achievement_hex**
    - Must contain valid hexadecimal characters (0-9, A-F)
    - Example: "1F"
4. Player Information:
    - Must store name as string in **player_name**
    - Cannot be empty
5. Player Stats Format:
    1. Calculate sum of all converted scores
    2. Must be stored as list in **player_stats**
    3. Must contain exactly **[player_name, total_score]**
    4. Example**: ["Steve", 250]**

3.2 Conversion Constraints

1. Mining Points Conversion:

- Convert string to integer using int()

- Store result in **mining_points**

2. Combat Score Conversion:
    - Convert float to integer using int()
    - Store result in combat_points
    - Must truncate decimal part
3. Achievement Bonus Conversion:
    - Convert hex to integer using int(x, 16)
    - Store result in **achievement_bonus**
4. Total Score Conversion:
    - Calculate sum of all converted scores
    - Store in **total_score**
    - Convert to string for display in **score_display**

3.3 Output Constraints

1. Display Format:
    - Show player name and all score components
    - Format total score as string in **score_display**
    - Store player record in **player_stats** list
    - Each score must be on a new line with proper labeling
    -

2. Required Output Format:
    - Show player name and all score components
    - Show "Mining Points: {value}"
    - Show "Combat Points: {value}"
    - Show "Achievement Bonus: {value}"

- Show "Total Score: {value}"

4. Template code Structure:

1. Type Conversion Functions:
    - convert_string_to_int()
    - convert_float_to_int()
    - convert_hex_to_int()
    - convert_score_to_string()
    - create_player_list()
2. Input Section:
    - Get mining score (string)
    - Get combat score (float)
    - Get achievement bonus (hex)
    - Get player name
3. Conversion Section:
    - Convert each score type
    - Calculate total score
    - Create player record
4. Output Section:
    - Display all score values
    - Show total score
    - Format player info correctly

5. Detailed Function Structure:

5.1 Data Conversion Functions

1. Write a Python function to convert string mining score to integer.

Define: **convert_string_to_int(mining_score)**

This method should:

- Accept a string parameter `mining_score` containing numeric characters
- Validate that the input is a string and contains only digits (validation is already provided)
- Convert the string to an integer using the `int()` function
- Return the converted integer value
- Handle the conversion properly without losing data
- Example: `convert_string_to_int("100")` should return `100`
- Example: `convert_string_to_int("0")` should return `0`
- The function will raise `ValueError` for invalid inputs (handled by existing validation)

2. Write a Python function to convert float combat score to integer.

Define: **convert_float_to_int(combat_score)**

This function should:

- Accept a float parameter `combat_score` representing combat accuracy
- Validate that the input is a float and non-negative (validation is already provided)
- Convert the float to an integer using the `int()` function (truncates decimal part)
- Return the converted integer value (decimal portion is discarded)
- Handle edge cases like `0.0` returning `0`
- Example: `convert_float_to_int(98.7)` should return `98`
- Example: `convert_float_to_int(1.0)` should return `1`
- The function will raise `ValueError` for invalid inputs (handled by existing validation)

3. Write a Python function to convert hexadecimal achievement score to integer.

Define: **convert_hex_to_int(achievement_hex)**

This function should:

- Accept a string parameter `achievement_hex` containing hexadecimal characters
- Validate that the input contains only valid hex characters 0-9, A-F, a-f (validation is already provided)
- Convert the hexadecimal string to integer using `int(achievement_hex, 16)`
- Return the converted integer value
- Handle both uppercase and lowercase hex characters
- Example: `convert_hex_to_int("1F")` should return `31`
- Example: `convert_hex_to_int("A")` should return `10`
- Example: `convert_hex_to_int("ff")` should return `255`
- The function will raise `ValueError` for invalid inputs (handled by existing validation)

5.2 Data Formatting Functions

4. Write a Python function to convert total score to string for display.

Define: **convert_score_to_string(total_score)**

This function should:

- Accept a numeric parameter `total_score` (int or float)
- Validate that the input is a number (validation is already provided)
- Convert the number to a string using the `str()` function
- Return the string representation of the score
- Maintain the exact numeric value in string format
- Example: `convert_score_to_string(150)` should return `"150"`
- Example: `convert_score_to_string(0)` should return `"0"`
- The function will raise `ValueError` for invalid inputs (handled by existing validation)

5. Write a Python function to create a player statistics list.

Define: **create_player_list(player_name, total_score**

This function should:

- Accept two parameters: `player_name` (string) and `total_score` (numeric)
- Validate that player_name is a non-empty string (validation is already provided)
- Create a list containing exactly two elements: `[player_name, total_score]`
- Return the list in the exact order: name first, score second
- Preserve the original data types within the list
- Example: `create_player_list("Steve", 100)` should return `["Steve", 100]`
- Example: `create_player_list("Alex", 250)` should return `["Alex", 250]`
- The function will raise `ValueError` for invalid inputs (handled by existing validation)

5.3 Main Program Implementation

6. Write the main program execution block.

Define: **if __name__ == "__main__"**

This function should:

- Print the welcome header and system information
- Use the exact variable names specified in the requirements:
  - `mining_score` (string input from user)
  - `mining_points` (converted integer)
  - `combat_score` (float from user input)
  - `combat_points` (converted integer)
  - `achievement_hex` (string input from user)
  - `achievement_bonus` (converted integer)
  - `total_score` (sum of all points)
  - `score_display` (total score as string)
  - `player_stats` (list with name and score)

6. Execution Steps to follow:

1. Run the program
2. Enter mining points when prompted
3. Enter combat accuracy score
4. Enter achievement bonus in hexadecimal
5. Enter Minecraft username
6. View complete score breakdown
7. Verify all conversions
8. Check final player statistics

Execution Steps to Follow:

- All actions like build, compile, running application, running test cases will be through Command Terminal.
- To open the command terminal the test takers, need to go to Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal
- This editor Auto Saves the code
- If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page) then you need to use CTRL+Shift+B -command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository. Else the code will not be available in the next login.
- These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.
- To setup environment:
- Pip install --upgrade typing_extensions
- To launch application: Python3 filename.py

- To run Test cases: In the terminal type pytest

- Before Final Submission also, you need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.

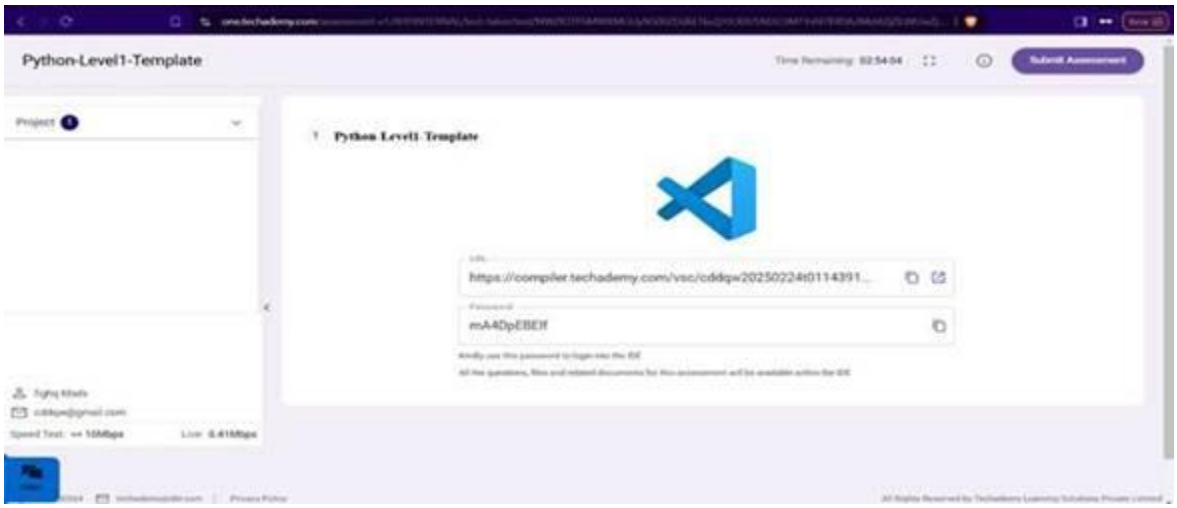
Screen shot to run the program

To run the application

Python3 filename.py

To run the testcase

type pytest in the terminal



- Once you are done with development and ready with submission, you may navigate to the previous tab and submit the workspace. It is mandatory to click on “Submit Assessment” after you are done with code.