Test Plan: BMI Program

Prepared by: Jerome Agoncillo

1. Introduction This test plan outlines the strategy for testing the BMI program to ensure it correctly calculates BMI, handles user inputs, and provides accurate results. The primary objective is to validate the program's accuracy and validity against valid and invalid inputs.

2. Testing Resources

- **Tester:** Jerome Agoncillo
- **Role:** Creator of the project and the test

3. Scope of Testing

- **In Scope:**
  - Validating user input for weight and height
  - Ensuring BMI calculation accuracy using:
    - **BMI = (weight (lbs) / height (in)^2) * 703**
  - Handling incorrect inputs without crashes
  - Testing edge cases, including extreme BMI values
- **Out of Scope:**
  - GUI/UI testing
  - Performance testing under high load

4. Testing Approaches

- **Manual Testing:** Validate user input handling through interactive prompts.
- **Negative Testing:** Provide invalid inputs to test error handling.
- **Types of Testing:**
  - **Functional Testing:** Verify correct BMI calculation and error handling.
  - **Usability Testing:** Ensure prompts and messages are clear and user-friendly.
  - **Boundary Testing:** Test extreme values to assess program stability.

5. Test Schedule

- **February 12-14:** Test Case Development
- **February 15-17:** Test Execution
- **February 18-21:** Bug Fixing & Retesting
- **February 21-23:** Final Review

6. Risks & Issues

- **Potential Risks:**
  - Invalid input handling may not be strong, leading to crashes.

    - Users may not follow input guidelines, causing unexpected behavior.
    - Calculation errors could arise from incorrect formula implementation.
- **Mitigation Strategies:**
    - Implement thorough input validation.
    - Provide clear user prompts and retry mechanisms.
    - Conduct rigorous testing across multiple scenarios.

7. Testing Technology: pytest is a popular testing framework for Python that simplifies writing and running tests with a concise framework. It supports fixtures, parameterized testing, and powerful plugins for scalable test automation. pytest is widely used for unit, integration, and functional testing due to its flexibility and ease of use. This is why I chose Pytest; also easier to read and understand.

8. Test Cases:

1. **test_userWeightAndHeight_validInput**

    - **What it tests**: This test verifies that the `userWeightAndHeight` function correctly processes valid user inputs.
    - **Inputs**:
        - `"150"` (valid weight)
        - `"5"` (valid height in feet)
        - `"10"` (valid height in inches)
        - `"c"` (continue input)
    - **Expected Output**:
        - Weight: `150`
        - Total Height: `(5 * 12) + 10 = 70` inches
2. **test_calculateBMI_validInput**

    - **What it tests**: Ensures that `calculateBMI` correctly calculates the BMI using valid inputs.
    - **Inputs**:
        - Weight: `150`
        - Height: `70` inches
    - **Expected Output**:
        - BMI: `round(150 / (70 ** 2) * 703, 1) = 21.5`
3. **test_userWeightAndHeight_invalidWeight**

    - **What it tests**: Verifies that `userWeightAndHeight` correctly prompts the user again when an invalid weight is entered.
    - **Inputs**:
        - `"abc"` (invalid weight)
        - `"150"` (valid weight after retry)

        - `"5"` (valid height in feet)
        - `"10"` (valid height in inches)
        - `"c"` (continue input)
    - **Expected Output**:
        - Weight: `150` (after retry)
4. **test_userWeightAndHeight_invalidHeightFt**

    - **What it tests**: Ensures the function handles invalid height (feet) input correctly and prompts the user again.
    - **Inputs**:
        - `"150"` (valid weight)
        - `"xyz"` (invalid height in feet)
        - `"5"` (valid height in feet after retry)
        - `"10"` (valid height in inches)
        - `"c"` (continue input)
    - **Expected Output**:
        - Total Height: `(5 * 12) + 10 = 70` inches (after retry)
5. **test_userWeightAndHeight_invalidHeightIn**

    - **What it tests**: Ensures the function correctly handles an invalid height (inches) input and prompts the user again.
    - **Inputs**:
        - `"150"` (valid weight)
        - `"5"` (valid height in feet)
        - `"ten"` (invalid height in inches)
        - `"10"` (valid height in inches after retry)
        - `"c"` (continue input)
    - **Expected Output**:
        - Total Height: `(5 * 12) + 10 = 70` inches (after retry)
6. **test_calculateBMI_zeroHeight**

    - **What it tests**: Ensures that `calculateBMI` raises an error when height is zero (to prevent division by zero).
    - **Inputs**:
        - Weight: `150`
        - Height: `0`
    - **Expected Output**:
        - A `ZeroDivisionError` should be raised.
7. **test_calculateBMI_negativeHeight**

- **What it tests**: Ensures that `calculateBMI` raises an error when height is negative.
- **Inputs**:
  - Weight: `150`
  - Height: `-70`
- **Expected Output**:
  - A `ValueError` should be raised.