Functions

Impure Functions:

1. **Initialize Character Stats**

    - Create a function `initializeCharacterStats` that initializes an empty array to store character stats. Each stat will eventually be an array with two elements: a string representing the character name and a number representing the character's strength. This function should set a global array variable `characterStats` to an empty array.

2. **Add Character Stat**

    - Write a function `addCharacterStat` that takes two parameters: `characterName` (a string) and `strength` (a number). This function should add a new sub-array to the `characterStats` array with the character name and strength.

3. **Find Character Strength**

    - Create a function `findCharacterStrength` that takes one parameter, `characterName` , and logs the strength of the character. If the character is not found, it should log `Character was not found` .

4. **Update Character Strength**

    - Write a function `updateCharacterStrength` that takes two parameters: `characterName` and `newStrength` . It should update the strength of the specified character. If the character does not exist in the list, it should print a message indicating so.

5. **Remove Character Stat**

    - Develop a function `removeCharacterStat` that takes one parameter, `characterName` , and removes the character from the `characterStats` array. If the character is not found, it should print a message indicating so.

6. **List All Characters and Strengths**
    - Create a function `listCharacters` that prints all characters and their strengths in the format "Character Name - Strength".

7. **Find Strongest Character**
    - Create a function `strongestCharacter` that finds and prints the name of the character with the highest strength. If there are no characters, it should print a message indicating so.

Pure Functions:

1. **Determine the Average Rating of Games:** Define a pure function `calculateAverageRating` that takes an array of game ratings as input and returns the average rating rounded to one decimal place.

- Input: `[8.5, 7.9, 6.3, 9.2, 8.1]`
- Output: `8.0`

2. **Extract Developer Names from Game Titles:**

- Develop a pure function named `extractDeveloperNames` that takes an array of game titles as input and extracts the names of developers from each title. The function should then return an array containing only the extracted developer names.
- Input Example: `['The Legend of Zelda: Breath of the Wild (2017) - Developed by Nintendo', 'Final Fantasy VII (1997) - Developed by Square Enix', 'Halo: Combat Evolved (2001) - Developed by Bungie']`
- Expected Output: `['Nintendo', 'Square Enix', 'Bungie']`

3. **Check if a Game Title Contains a Specific Word:**
    - Implement a pure function `containsWordInTitle` that takes a game title and a keyword as input and returns `true` if the title contains the keyword, otherwise `false`.
    - Input: `('Final Fantasy VII', 'Fantasy')`
    - Output: `true`

4. **Check if All Game Titles Start with the Same Letter:**

   - Define a pure function `allTitlesStartWithSameLetter` that takes an array of game titles as input and returns `true` if all titles start with the same letter, otherwise `false`.
   - Input: `['Assassin's Creed', 'Age of Empires', 'Animal Crossing']`
   - Output: `true`

5. **Concatenate All Game Genres into a Single String:**

   - Create a pure function `concatenateGenres` that takes an array of game genres as input and returns a single string containing all the genres concatenated together without using the `join()` method.
   - Input: `['RPG', 'Action', 'Adventure']`
   - Output: `'RPG, Action, Adventure'`

6. **Count Battles in XP Range:**

- Design a pure function `countInXPRange` that takes an array of experience points and two numbers representing the lower and upper bounds of an XP range (inclusive). The function returns the count of battles within that XP range.
- **Input**: `([50, 75, 100, 30, 20, 100], 75, 100)`
- **Expected Output**: `3`

7. **Find Games Developed by a Specific Developer:**

   - Define a pure function named `findGamesByDeveloper` that takes two arguments: an array of game titles and an array of corresponding developers, where each developer corresponds to the game title at the same index. This function should filter out and return a new array containing only the titles of games developed by a specific developer.
   - Input Example: `['The Legend of Zelda', 'Final Fantasy', 'Halo: Combat Evolved'], ['Nintendo', 'Square Enix', 'Bungie'], 'Nintendo'`

    - Expected Output: `['The Legend of Zelda']`

8. **Capitalize the First Letter of Each Game Title:**

    - Define a pure function `capitalizeFirstLetter` that takes an array of game titles as input and returns a new array where the first letter of each title is capitalized. **Note: Do not modify the original array.**
    - Input: `['the legend of zelda', 'final fantasy', 'halo: combat evolved']`
    - Output: `['The legend of zelda', 'Final fantasy', 'Halo: combat evolved']`

9. **Filter Unique XP Values:**

- Develop a pure function `uniqueXP` that takes an array of experience points and returns a new array of XP values without duplicates, preserving their original order in the input array.
- **Input**: `[100, 50, 75, 50, 100, 75, 100]`
- **Expected Output**: `[100, 50, 75]`

10. **Sort Game Titles Alphabetically:** - Write a pure function `sortTitlesAlphabetically` that takes an array of game titles as input and returns a new array with the titles sorted alphabetically. **Note: Do not modify the original array.** - Input: `['The Legend of Zelda', 'Final Fantasy', 'Halo: Combat Evolved']`
    - Output: `['Final Fantasy', 'Halo: Combat Evolved', 'The Legend of Zelda']`

11. **Filter XP by Multiple Criteria:**

- Write a pure function `filterXPByCriteria` that takes an array of XP and multiple criteria (e.g., greater than a value, less than a value), and returns a new array of XP that meet all criteria.
- **Input**: `([10, 20, 30, 40, 50], greaterThan=20, lessThan=50)`

- **Expected Output**: `[30, 40]`

Bonus Exercises

1. **Sort Experience Points:**

- Develop a pure function `sortXP` that takes an array of experience points and returns a new array with the XP values sorted from lowest to highest. Implement the sorting algorithm using loops and without the `sort` method. Ensure it does not modify the input array.
- **Input**: `[100, 50, 75, 25, 10]`
- **Expected Output**: `[10, 25, 50, 75, 100]`

2. **Average XP of Top N Battles:**

- Develop a pure function `averageOfTopNXP` that takes an array of experience points and an integer `N`. It should return the average XP of the top `N` XP values. You can use the function you created in exercise 1 in this section in order to sort the arrays.
- **Input**: `([50, 10, 100, 75, 25], 3)`
- **Expected Output**: `75` (The top 3 XP values are 100, 75, and 50)

3. **Normalize Experience Points:**

- Create a pure function `normalizeXP` that takes an array of XP (0-100) and scales them to a new range (e.g., 0-10), returning a new array of normalized experience points.
- **Input**: `([0, 25, 50, 75, 100], 0, 10)`
- **Expected Output**: `[0, 2.5, 5, 7.5, 10]`