|  |  |  |
| --- | --- | --- |
| **Feature** | **majorityElement Function** | **MajorityElement Class** |
| **Structure** | A standalone function that processes an array directly. | A class-based approach with a method (findMajorityElement) for finding the majority element. |
| **State Management** | No state is maintained, all variables (map, value1, s, etc.) are local to the function. | The state (like the frequency map) is encapsulated within the class and reusable through different methods. |
| **Constructor** | No constructor is used, values are passed directly to the function. | No constructor in this specific implementation, but the class structure makes it easy to add if needed. |
| **Modularity** | Less modular. You would have to rewrite the logic or repeat the code if you wanted to reuse it. | More modular. Can reuse the class and add new methods for extended functionality. |
| **Return Value** | Returns all elements that have the highest frequency as an array of [element, count]. | Returns a single majority element that appears the most in the array. |
| **Data Processing** | Uses a Map to count occurrences, then finds all elements with the maximum count. | Uses a Map to count occurrences, then finds the element with the highest count (majority element). |
| **Output** | Returns an array of the elements with the highest frequency, not limited to just one. | Returns a single majority element with the highest count. |
| **Flexibility** | Can handle multiple "majority" elements if they have the same frequency. | Designed to handle only one majority element, assuming there’s a single element with the highest count. |
| **Use Case** | Suitable when you want to find **all** elements with the maximum frequency, even if there are ties. | Suitable when there’s one majority element that dominates in frequency. |
| **Method/Function Call** | Called directly with the input array, like majorityElement([1, 2, 3, 3, 4, 2, 3, 2, 4, 4]). | Called on an object, like majorityElementFinder.printMajorityElement(inputArray). |
| **Ease of Understanding** | Easier to understand for beginners, as it’s a simple function with no additional structure. | More object-oriented, potentially harder to understand for beginners, but offers better reusability. |
| **Scalability** | Harder to scale, as each function would need to be standalone or rewritten. | Easier to scale, as you can add more methods to the class or reuse the class in larger projects. |