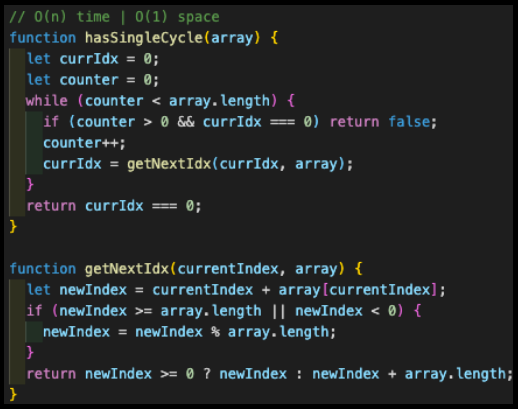
Input: An array of integers //each integer represents a jump of its value in the array. Eg. integer 2 represents 2 index jump forward.
// If jump exceeds array bounds, it jumps to the other side.

array = [2, 3, 1, -4, -4, 2]

Output: Boolean //representing whether the jumps form a single cycle. A single cycle occurs if, starting at any index in the array and following the jumps, every element in the array is visited exactly once before landing back on the starting index

true



```javascript
// O(n) time | O(1) space
function hasSingleCycle(array) {
  let currIdx = 0;
  let counter = 0;
  while (counter < array.length) {
    if (counter > 0 && currIdx === 0) return false;
    counter++;
    currIdx = getNextIdx(currIdx, array);
  }
  return currIdx === 0;
}

function getNextIdx(currentIndex, array) {
  let newIndex = currentIndex + array[currentIndex];
  if (newIndex >= array.length || newIndex < 0) {
    newIndex = newIndex % array.length;
  }
  return newIndex >= 0 ? newIndex : newIndex + array.length;
}
```

Idea: We know how many elements are in our array. We also know what index we start at.
If we arrive at the starting index before our counter reaches the array length, we can return false since we would have visited the node twice
If at the end we do not end at our starting index that means we did not do a single cycle
getNextIdx handles updating our current index and handling the edge cases (array out of bounds)

Time: O(n) (where n is the length of the array) since we are iterating through n elements and every other operation is O(1)

Space: O(1) since we are not storing any more values as n grows.