```javascript
/* jshint esversion: 6 */
// 1. Calculate the factorial of a number. The factorial of a non-negative integer n,
// denoted by n!, is the product of all positive integers less than or equal to n.
// Example: 5! = 5 x 4 x 3 x 2 x 1 = 120
// factorial(5); // 120
let factorial = function ( n ) {
    // base cases
    if ( n < 0 || n === undefined ) {
        return null;
    }
    if ( n <= 1 ) {
        return 1;
    }
    // recursive case
    return n * factorial( n - 1 );
};
//-----------------------------------------------
// 2. Compute the sum of an array of integers.
// sum([1,2,3,4,5,6]); // 21
let sum = function ( array ) {
    // base case
    if ( !array.length ) {
        return 0;
    }
    // recursive case
    return array[ 0 ] + sum( array.slice( 1 ) );
};
//-----------------------------------------------
// 3. Sum all numbers in an array containing nested arrays.
// arraySum([1,[2,3],[[4]],5]); // 15
let arraySum = function ( array ) {
    return array.reduce( ( sum, el ) => {
        // base case: add integer element to sum
        // recursive case: call arraySum on non-integer element
        return sum + ( Array.isArray( el ) ? arraySum( el ) : el )
    }, 0 );
};
//-----------------------------------------------
// 4. Check if a number is even.
let isEven = function ( n ) {
    // base cases
    if ( n === 0 ) {
        return true;
    }
    if ( n === 1 || n === -1 ) {
        return false;
    }
    // recursive case
    return isEven( n > 0 ? n - 2 : n + 2 );
};
//-----------------------------------------------
// 5. Sum all integers below a given integer.
// sumBelow(10); // 45
// sumBelow(7); // 21
let sumBelow = function ( n ) {
    // base case
    if ( n === 0 ) {
        return 0;
    }
    // recursive case
    n = n > 0 ? n - 1 : n + 1;
    return n + sumBelow( n );
};
//-----------------------------------------------
// 6. Get the integers within a range (x, y).
// range(2,9); // [3,4,5,6,7,8]
let range = function ( x, y ) {
    // base case
    if ( y - x === 1 || y - x === 0 ) {
        return [];
    }
    // recursive case
    y = y > x ? y - 1 : y + 1
    return y === x ? [] : range( x, y ).concat( y );
};
//-----------------------------------------------
// 7. Compute the exponent of a number.
// The exponent of a number says how many times the base number is used as a factor.
// 8^2 = 8 x 8 = 64. Here, 8 is the base and 2 is the exponent.
// exponent(4,3); // 64
// https://www.khanacademy.org/computing/computer-science/algorithms/recursive-algorithms/a/computing-powers-of-a-number
let exponent = function ( base, exp ) {
    // base case
    if ( exp === 0 ) {
        return 1;
    }
    // recursive cases!
    return exp > 0 ? base * exponent( base, exp - 1 ) : 1 / ( base * exponent( base, -1 * exp - 1 ) );
};
//-----------------------------------------------
// 8. Determine if a number is a power of two.
// powerOfTwo(1); // true
// powerOfTwo(16); // true
// powerOfTwo(10); // false
let powerOfTwo = function ( n ) {
    // base cases
    if ( n === 1 ) {
        return true;
    }
    if ( n === 0 || n % 2 === 1 ) {
        return false;
    }
    // recursive case
    return powerOfTwo( n / 2 );
};
//-----------------------------------------------
// 9. Write a function that reverses a string.
let reverse = function ( string ) {
    // base case: resolve to empty string
    // recursive case: call reverse on substring
    return string === '' ? '' : reverse( string.substr( 1 ) ) + string.charAt( 0 );
};
//-----------------------------------------------
// 10. Write a function that determines if a string is a palindrome.
let palindrome = function ( string ) {
    // base cases
    if ( string === '' || string.length === 1 ) {
        return true;
    }
    if ( string[ 0 ].toLowerCase() !== string.slice( -1 ).toLowerCase() ) {
        return false;
    }
    // recursive case
    return palindrome( string.substr( 1, string.length - 2 ) );
};
//-----------------------------------------------
// 11. Write a function that returns the remainder of x divided by y without using the
// modulo (%) operator.
// modulo(5,2) // 1
// modulo(17,5) // 2
// modulo(22,6) // 4
let modulo = function ( x, y ) {
    if ( y === 0 ) {
        return NaN;
    }
    if ( x < 0 && y < 0 ) {
        if ( x > y ) {
            return x;
        }
    } else if ( ( x < 0 && y > 0 ) || ( x > 0 && y < 0 ) ) {
        if ( -x < y ) {
            return x;
        }
        return modulo( x + y, y );
    } else {
        if ( x < y ) {
            return x;
        }
    }
    return modulo( x - y, y );
};
//---------------------------------------------
// 12. Write a function that multiplies two numbers without using the * operator or
// Math methods.
let multiply = function ( x, y ) {
    if ( x === 0 || y === 0 ) {
        return 0;
    } else if ( y < 0 ) {
        return -x + multiply( x, y + 1 );
    } else {
        return x + multiply( x, y - 1 );
    }
};
//---------------------------------------------
// 13. Write a function that divides two numbers without using the / operator or
// Math methods to arrive at an approximate quotient (ignore decimal endings).
let divide = function ( x, y ) {
    if ( y === 0 ) {
        return NaN;
    }
    if ( x === 0 ) {
        return 0;
    }
    if ( x < 0 && y > 0 && -x < y || x < -y ) {
        return 0;
    }
    if ( x > 0 && y > 0 && x < y ) {
        return 0;
    }
    if ( x > 0 && y > 0 ) {
        return 1 + divide( x - y, y );
    } else {
        return -1 + divide( x + y, y );
    }
}
//---------------------------------------------
// 14. Find the greatest common divisor (gcd) of two positive numbers. The GCD of two
// integers is the greatest integer that divides both x and y with no remainder.
// gcd(4,36); // 4
// http://www.cse.wustl.edu/~kjg/cse131/Notes/Recursion/recursion.html
// https://www.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/the-euclidean-algorithm
let gcd = function ( x, y ) {
    // base cases
    if ( x < 0 || y < 0 ) {
        return null;
    }
    if ( y % x === 0 ) {
        return x;
    }
    // recursive cases
    return x > y ? gcd( y, x ) : gcd( x, y % x );
};
//---------------------------------------------
// 15. Write a function that compares each character of two strings and returns true if
// both are identical.
// compareStr('house', 'houses') // false
// compareStr('tomato', 'tomato') // true
let compareStr = function ( str1, str2 ) {
    // base cases
    if ( str1 === '' && str2 === '' ) {
        return true;
    }
    if ( str1.charAt( 0 ) !== str2.charAt( 0 ) ) {
        return false;
    }
    // recursive case
    return compareStr( str1.substr( 1 ), str2.substr( 1 ) );
};
//---------------------------------------------
```

```
// 16. Write a function that accepts a string and creates an array where each letter
// occupies an index of the array.
let createArray = function ( str ) {
    // base case: resolve to array containing first character of string
    // recursive case: concatenate results of calling createArray on substrings
    return str.length === 1 ? [ str.charAt( 0 ) ] : [ str.charAt( 0 ) ].concat( createArray( str.substr( 1 ) ) );
};
//----------------------------------------------
// 17. Reverse the order of an array
let reverseArr = function ( array ) {
    // base case: resolve to (original) empty array
    // recursive case: call reverseArray on sliced array and concatenate with first element
    return !array.length ? array : reverseArr( array.slice( 1 ) ).concat( array[ 0 ] );
};
//----------------------------------------------
// 18. Create a new array with a given value and length.
// buildList(0,5) // [0,0,0,0,0]
// buildList(7,3) // [7,7,7]
let buildList = function ( value, length ) {
    // base case: resolve to empty array
    // recursive case: concatenate results of calling buildList on value and decremented length
    return length === 0 ? [] : [ value ].concat( buildList( value, length - 1 ) );
};
//----------------------------------------------
// 19. Implement FizzBuzz. Given integer n, return an array of the string representations of 1 to n.
// For multiples of three, output 'Fizz' instead of the number.
// For multiples of five, output 'Buzz' instead of the number.
// For numbers which are multiples of both three and five, output "FizzBuzz" instead of the number.
// fizzBuzz(5) // ['1','2','Fizz','4','Buzz']
let fizzBuzz = function ( n ) {
    let results = [];
    let val = n;
    // base case
    if ( n === 0 ) {
        return results;
    }
    // recursive cases
    if ( n % 3 === 0 && n % 5 !== 0 ) {
        val = 'Fizz';
    }
    if ( n % 3 !== 0 && n % 5 === 0 ) {
        val = 'Buzz';
    }
    if ( n % 3 === 0 && n % 5 === 0 ) {
        val = 'FizzBuzz';
    }
    results.push( val.toString() );
    return fizzBuzz( n - 1 ).concat( results );
};
//----------------------------------------------
// 20. Count the occurence of a value in a list.
// countOccurrence([2,7,4,4,1,4], 4) // 3
// countOccurrence([2,'banana',4,4,1,'banana'], 'banana') // 2
let countOccurrence = function ( array, value ) {
    // base case
    if ( array.length === 0 ) {
        return 0;
    }
    // recursive case
    let increment;
    if (array[ 0 ] === value) {
        increment = 1;
    } else {
        increment = 0;
    }
    return increment + countOccurrence( array.slice( 1 ), value );
};
// 21. Write a recursive version of map.
// rMap([1,2,3], timesTwo); // [2,4,6]
//--------------------------------Without Recursion------
// Write a function using fat arrow syntax named`arrowMyMap` that accepts an array
// and a callback as arguments.The function will return an array of new elements
// obtained by calling the callback on each element of the array, passing in the
```

```javascript
// element.Assign the below function to a letiable using the const keyword.
//
//     Do not use the built in Array#map - use Array#forEach for iteration.
//
//
//         Examples:
//     let result1 = arrowMyMap( [ 100, 25, 81, 64 ], Math.sqrt );
// console.log( result1 );   // [ 10, 5, 9, 8 ]
//
// const yell = el => el.toUpperCase() + '!'
//
// let result2 = arrowMyMap( [ 'run', 'Forrest' ], yell );
// console.log( result2 );   // [ 'RUN!', 'FORREST!' ]
//
// ********************************************************************** /
//
// const arrowMyMap = ( array, cb ) => {
//     let mapped = [];
//
//     array.forEach( el => mapped.push( cb( el ) ) );
//     return mapped;
// };
//-------------------------------With Recursion------
let rMap = function ( array, callback ) {
    if ( array.length === 0 ) return [];
    let list = rMap( array.slice( 1, array.length ), callback );
    list.unshift( callback( array[ 0 ] ) );
    return list;
};
// 22. Write a function that counts the number of times a key occurs in an object.
// let obj = {'e':{'x':'y'},'t':{'r':{'e':'r'},'p':{'y':'r'}},'y':'e'};
// countKeysInObj(obj, 'r') // 1
// countKeysInObj(obj, 'e') // 2
let countKeysInObj = function ( obj, key ) {
    let count = 0;
    for ( let prop in obj ) {
        if ( prop === key ) {
            count++;
        }
        if ( obj[ prop ] instanceof Object ) {
            count += countKeysInObj( obj[ prop ], key );
        }
    }
    return count;
};
// 23. Write a function that counts the number of times a value occurs in an object.
// let obj = {'e':{'x':'y'},'t':{'r':{'e':'r'},'p':{'y':'r'}},'y':'e'};
// countValuesInObj(obj, 'r') // 2
// countValuesInObj(obj, 'e') // 1
let countValuesInObj = function ( obj, value ) {
    let count = 0;
    for ( let prop in obj ) {
        if ( obj[ prop ] === value ) count++;
        if ( obj[ prop ] instanceof Object ) {
            count += countValuesInObj( obj[ prop ], value );
        }
    }
    return count;
};
// 24. Find all keys in an object (and nested objects) by a provided name and rename
// them to a provided new name while preserving the value stored at that key.
let replaceKeysInObj = function ( obj, oldKey, newKey ) {
    for ( let prop in obj ) {
        if ( prop === oldKey ) {
            obj[ newKey ] = obj[ prop ];
            delete obj[ prop ];
        }
        if ( obj[ prop ] instanceof Object ) {
            replaceKeysInObj( obj[ prop ], oldKey, newKey );
        }
    }
    return obj;
};
// 25. Get the first n Fibonacci numbers. In the Fibonacci sequence, each subsequent
// number is the sum of the previous two.
// Example: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34.....
// fibonacci(5); // [0,1,1,2,3,5]
// Note: The 0 is not counted.
let fibonacci = function ( n ) {
    if ( n <= 0 ) return null;
    if ( n === 1 ) return [ 0, 1 ];
    let list = fibonacci( n - 1 );
    if ( list[ n ] !== undefined ) return list[ n ];
    list[ n ] = list[ n - 1 ] + list[ n - 2 ];
    return list;
};
// 26. Return the Fibonacci number located at index n of the Fibonacci sequence.
// [0,1,1,2,3,5,8,13,21]
// nthFibo(5); // 5
// nthFibo(7); // 13
// nthFibo(3); // 2
let nthFibo = function ( n ) {
    if ( n < 0 ) return null;
    if ( n === 0 ) return 0;
    if ( n === 1 ) return 1;
    return nthFibo( n - 1 ) + nthFibo( n - 2 );
};
// 27. Given an array of words, return a new array containing each word capitalized.
// let words = ['i', 'am', 'learning', 'recursion'];
// capitalizedWords(words); // ['I', 'AM', 'LEARNING', 'RECURSION']
let capitalizeWords = function ( array ) {
    if ( array.length === 0 ) return [];
    let list = capitalizeWords( array.slice( 1, array.length ) );
    list.unshift( array[ 0 ].toUpperCase() );
    return list;
};
// 28. Given an array of strings, capitalize the first letter of each index.
// capitalizeFirst(['car','poop','banana']); // ['Car','Poop','Banana']
let capitalizeFirst = function ( array ) {
    if ( array.length === 0 ) return [];
    let list = capitalizeFirst( array.slice( 1, array.length ) );
    list.unshift( array[ 0 ][ 0 ].toUpperCase() + array[ 0 ].substring( 1 ) );
    return list;
};
// 29. Return the sum of all even numbers in an object containing nested objects.
// let obj1 = {
//   a: 2,
//   b: {b: 2, bb: {b: 3, bb: {b: 2}}},
//   c: {c: {c: 2}, cc: 'ball', ccc: 5},
//   d: 1,
//   e: {e: {e: 2}, ee: 'car'}
// };
// nestedEvenSum(obj1); // 10
let nestedEvenSum = function ( obj ) {
    let sum = 0;
    for ( let prop in obj ) {
        if ( obj[ prop ] % 2 === 0 ) sum += obj[ prop ];
        if ( obj[ prop ] instanceof Object ) sum += nestedEvenSum( obj[ prop ] );
    }
    return sum;
};
// 30. Flatten an array containing nested arrays.
// flatten([1,[2],[3,[[4]]],5]); // [1,2,3,4,5]
let flatten = function ( array ) {
    let list = [];
    if ( array.length === 0 ) return [];
    for ( let i = 0; i < array.length; i++ ) {
        if ( array[ i ] instanceof Array ) {
            list.push( ...flatten( array[ i ] ) );
        } else {
            list.push( array[ i ] );
        }
    }
    return list;
};
// 31. Given a string, return an object containing tallies of each letter.
// letterTally('potato'); // {p:1, o:2, t:2, a:1}
let letterTally = function ( str, obj = {} ) {
```

```javascript
    if ( str.length === 0 ) return obj;
    letterTally( str.substring( 1 ), obj );
    if ( obj[ str[ 0 ] ] === undefined ) {
        obj[ str[ 0 ] ] = 1;
    } else {
        obj[ str[ 0 ] ] += 1;
    }
    return obj;
};
// 32. Eliminate consecutive duplicates in a list. If the list contains repeated
// elements they should be replaced with a single copy of the element. The order of the
// elements should not be changed.
// compress([1,2,2,3,4,4,5,5,5]) // [1,2,3,4,5]
// compress([1,2,2,3,4,4,2,5,5,5,4,4]) // [1,2,3,4,2,5,4]
let compress = function ( list ) {
    if ( list.length === 0 ) return [];
    let res = compress( list.slice( 1 ) );
    if ( list[ 0 ] !== res[ 0 ] ) {
        res.unshift( list[ 0 ] );
    }
    return res;
};
// 33. Augument every element in a list with a new value where each element is an array
// itself.
// augmentElements([[],[3],[7]], 5); // [[5],[3,5],[7,5]]
let augmentElements = function ( array, aug ) {
    if ( array.length === 0 ) return [];
    let list = augmentElements( array.slice( 1 ), aug );
    array[ 0 ].push( aug );
    list.unshift( array[ 0 ] );
    return list;
};
// 34. Reduce a series of zeroes to a single 0.
// minimizeZeroes([2,0,0,0,1,4]) // [2,0,1,4]
// minimizeZeroes([2,0,0,0,1,0,0,4]) // [2,0,1,0,4]
let minimizeZeroes = function ( array ) {
    if ( array.length === 0 ) return [];
    let list = minimizeZeroes( array.slice( 1 ) );
    if ( ( array[ 0 ] === 0 ^ list[ 0 ] === 0 ) || array[ 0 ] !== 0 ) {
        list.unshift( array[ 0 ] );
    }
    return list;
};
// 35. Alternate the numbers in an array between positive and negative regardless of
// their original sign. The first number in the index always needs to be positive.
// alternateSign([2,7,8,3,1,4]) // [2,-7,8,-3,1,-4]
// alternateSign([-2,-7,8,3,-1,4]) // [2,-7,8,-3,1,-4]
let alternateSign = function ( array ) {
    if ( array.length === 0 ) return [];
    let list = alternateSign( array.slice( 0, array.length - 1 ) );
    let lng = array.length;
    if ( lng % 2 === 0 ) {
        if ( array[ lng - 1 ] > 0 ) {
            array[ lng - 1 ] = -array[ lng - 1 ];
        }
    } else {
        if ( array[ lng - 1 ] < 0 ) {
            array[ lng - 1 ] = -array[ lng - 1 ];
        }
    }
    list.push( array[ lng - 1 ] );
    return list;
};
// 36. Given a string, return a string with digits converted to their word equivalent.
// Assume all numbers are single digits (less than 10).
// numToText("I have 5 dogs and 6 ponies"); // "I have five dogs and six ponies"
let numToText = function ( str ) {
    if ( str.length === 0 ) return '';
    let tempStr = numToText( str.substring( 0, str.length - 1 ) );
    let replace;
    switch ( str[ str.length - 1 ] ) {
        case '1': replace = 'one';
            break;
        case '2': replace = 'two';
```

```javascript
            break;
        case '3': replace = 'three';
            break;
        case '4': replace = 'four';
            break;
        case '5': replace = 'five';
            break;
        case '6': replace = 'six';
            break;
        case '7': replace = 'seven';
            break;
        case '8': replace = 'eight';
            break;
        case '9': replace = 'nine';
            break;
        default: replace = str[ str.length - 1 ];
            break;
    }
    return tempStr + replace;
};
// *** EXTRA CREDIT ***
// 37. Return the number of times a tag occurs in the DOM.
let tagCount = function ( tag, node ) {
};
// 38. Write a function for binary search.
// let array = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15];
// binarySearch(array, 5) // 5
// https://www.khanacademy.org/computing/computer-science/algorithms/binary-search/a/binary-search
let binarySearch = function ( array, target, min = 0, max = array.length - 1 ) {
    if ( array.length === 0 ) return null;
    if ( min > max ) return null;
    let mid = Math.floor( ( max - min ) / 2 ) + min;
    if ( array[ mid ] < target ) {
        return binarySearch( array, target, mid + 1, max );
    } else if ( array[ mid ] > target ) {
        return binarySearch( array, target, min, mid - 1 );
    } else {
        return mid;
    }
};
// 39. Write a merge sort function.
// mergeSort([34,7,23,32,5,62]) // [5,7,23,32,34,62]
// https://www.khanacademy.org/computing/computer-science/algorithms/merge-sort/a/divide-and-conquer-algorithms
let mergeSort = function ( array ) {
    if ( array.length === 0 || array.length === 1 ) return array;
    let mid = Math.floor( array.length / 2 );
    let left = array.slice( 0, mid );
    let right = array.slice( mid, array.length );
    return merge( mergeSort( left ), mergeSort( right ) );
};
function merge ( left, right ) {
    let res = [];
    let l = 0;
    let r = 0;
    while ( l < left.length && r < right.length ) {
        if ( left[ l ] < right[ r ] ) {
            res.push( left[ l++ ] );
        } else {
            res.push( right[ r++ ] );
        }
    }
    return res.concat( left.slice( l ) ).concat( right.slice( r ) );;
}
// 40. Deeply clone objects and arrays.
// let obj1 = {a:1,b:{bb:{bbb:2}},c:3};
// let obj2 = clone(obj1);
// console.log(obj2); // {a:1,b:{bb:{bbb:2}},c:3}
// obj1 === obj2 // false
let clone = function ( input ) {
    let res;
    if ( Array.isArray( input ) ) {
        res = [];
        if ( input.length === 0 ) return [];
        for ( let i = 0; i < input.length; i++ ) {
            if ( input[ i ] instanceof Object ) {
                res.push( clone( input[ i ] ) );
            } else {
                res.push( input[ i ] );
            }
        }
    } else {
        if ( input === null ) return {};
        res = {};
        for ( let prop in input ) {
            if ( input[ prop ] instanceof Object ) {
                res[ prop ] = clone( input[ prop ] );
            } else {
                res[ prop ] = input[ prop ];
            }
        }
    }
    return res;
};
```