SENTENCE REVERSAL

```
/**
 * @param {string} str
 * @returns {string}
 */
var reverseWords = function(str) {
   return str.trim().split(/\s+/).reverse().join(" ")
};

var reverseWords = function(str) {
   let arr = str.split(' ');
   let ans = '';

   for(let i = arr.length-1; i > -1; i--)
      {
         if(arr[i] !== '')
         {
            ans += arr[i] + ' ';
         }
      }
   return ans.trim();
};
```

SQRT OF X

```
var mySqrt = function(x) {
   if (x === null || x === undefined){
      return -1
   }
   var left = 0
   var right = Math.floor(x/2) + 1
   var mid

   while (left <= right){
      mid = Math.floor((left + right)/2)
      if (mid * mid > x){
         right = mid - 1
      } else if (mid * mid < x){
         left = mid + 1
      } else {
         return mid
      }
   }
   return mid * mid <=x? mid: mid-1;
   // return right
};
```

STRING TO INTEGER ATOI

```
/**
 * @param {string} str
 * @return {number}
 */
var myAtoi = function(str) {
    let i = 0;
    let res = 0;
    let isNegative = false;

    str = str.trim()

    // 2. Optional +/-
    const maybeSign = str[i];
    if (maybeSign === '+' || maybeSign === '-') {
        isNegative = maybeSign === '-';
        i += 1;
    }

// 3. Process numbers and stop once an invalid character is found
    for (; i < str.length; i += 1) {
        const code = str.charCodeAt(i) - 48; // '0' is 48
        if (code < 0 || code > 9) {
            break;
        }
        res *= 10;
        res += code;
    }

    if (isNegative) {
        res = -res;
    }
    return Math.max(-(2**31), Math.min(2**31 - 1, res));
};
```

STRSTR

/*
Implement strStr().

Return the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack.

Example 1:

Input: haystack = "hello", needle = "ll"
Output: 2
Example 2:

Input: haystack = "aaaaa", needle = "bba"
Output: -1
Clarification:

What should we return when needle is an empty string? This is a great question to ask during an interview.

For the purpose of this problem, we will return 0 when needle is an empty string. This is consistent to C's strstr() and Java's indexOf().
*/

// Better Implementation

```
var strStr = function(haystack, needle) {
   if (!needle){
      return 0
   }
   for (let i = 0; i < haystack.length; i++){
      if (haystack.slice(i, i + needle.length) === needle){
         return i
      }
   }
   return -1
};
```

SUM BIT WISE

```
/*
Calculate the sum of two integers a and b, but you are not allowed to use the operator + and -.

Example 1:

Input: a = 1, b = 2
Output: 3
Example 2:

Input: a = -2, b = 3
Output: 1
*/

/**
 * @param {number} a
 * @param {number} b
 * @return {number}
 * Explanation:
 * let's say a and b are both one-bit variable
 * There are only four bit sum condition: (1, 1), (1, 0), (0, 1), (0, 0)
 * Their bit sum is (10), (1), (1), (0) in that sequence
 * for the condition (1, 0), (0, 1), (0, 0), we can use a^b
 * and for the condition (1, 1), we will need to use a&b<<1 (a&b make sure it's both 1, and left shift
for carry)
 * the final results are the combination of these two equations
 * Now we have: a + b = a&b<<1 + a^b
 */
var getSum = function(a, b) {
   if (a === 0){
      return b
   }

   if (b === 0){
      return a
   }

   while( b != 0){
       // imagine newA = a^b
      // imagine newB = a&b<<1
      // we keep looping until b goes to zero
      // b will eventually go to zero since b is keeping shifting to the left, and b is result of (a&b)
      let carry = (a&b) << 1
      a = a ^ b
      b = carry
   }
   return a
};
```

SWAP NODES IN PAIR

```
/*
Given a linked list, swap every two adjacent nodes and return its head.

Example:

Given 1->2->3->4, you should return the list as 2->1->4->3.
Note:

Your algorithm should use only constant extra space.
You may not modify the values in the list's nodes, only nodes itself may be changed.
*/

/**
 * Definition for singly-linked list.
 * function ListNode(val) {
 *     this.val = val;
 *     this.next = null;
 * }
 */
/**
 * @param {ListNode} head
 * @return {ListNode}
 */
var swapPairs = function(head) {
   let dummyHead = new ListNode(0)
   dummyHead.next = head

    let p = head;
    let prev = dummyHead;

    while (p && p.next){
       let q = p.next
       let r = p.next.next

       prev.next = q
       q.next = p
       p.next = r

       prev = p
       p = r
    }

   return dummyHead.next
 };
```

STRING COMPRESSION

```
/**
CTCI - 1.6
String Compression:
Implement a method to perform basic string compression using the counts of repeated characters.
For example, the string aabcccccaaa would become a2b1c5a3.
If the "compressed" string would not become smaller than the original string,
your method should return the original string. You can assume the string
has only uppercase and lowercase letters (a - z).
I: string
O: compressed string
C: optimize
E: empty string, compressed string that's same length as the original string
*/

//time complexity: linear
//space complexity: constant


function stringCompression(s) {
   let count = 1
   let out = ''
   for (let i = 0; i < s.length; i++){
      let curChar = s[i];
      let nextChar = s[i + 1];
      if (curChar === nextChar){
         count++
      } else {
         out = out + curChar + String(count);
         count = 1;
      }
   }
   return out.length < s.length ? out : s;
}

console.log(
   stringCompression('aabcccccaaa') === 'a2b1c5a3',
   stringCompression('aa') === 'aa',
   stringCompression('aaAAaa') === 'aaAAaa',
   stringCompression('aaaAAaa') === 'a3A2a2',
   stringCompression('') === ''
 );
```

SYMMMETRIC TREE

```
/*
Given a binary tree, check whether it is a mirror of itself (ie, symmetric around its center).

For example, this binary tree [1,2,2,3,4,4,3] is symmetric:

   1
  / \
 2   2
/ \ / \
3  4 4  3
But the following [1,2,2,null,3,null,3] is not:
   1
  / \
 2   2
  \   \
  3    3
Note:
Bonus points if you could solve it both recursively and iteratively.
*/
// Recursive approach
var isSymmetric = function(root) {
   if(!root){
      return true;
   }
   return check(root.left, root.right);
};


var check = function(left, right){
   if(!left && !right){
      return true;
   }
   if(!left || !right){
      return false;
   }
   if (left.val !== right.val) {
      return false;
   }
   return check(left.left, right.right) && check(right.left, left.right);
};

// Iterative approach

var isSymmetric = function(root) {
   let queue = []

   queue.unshift(root)
```

```
    queue.unshift(root)

    while (queue.length){
        let left = queue.shift()
        let right = queue.shift()
        if (!left && !right) {
            continue;
        }
        if (!left || !right) {
            return false;
        }
        if (left.val !== right.val){
            return false;
        }
        queue.unshift(left.left)
        queue.unshift(right.right)
        queue.unshift(left.right)
        queue.unshift(right.left)
    }
    return true
}
```

TWO SUM

```
/**
 * @param {number[]} nums
 * @param {number} target
 * @return {number[]}
 */
var twoSum = function(nums, target) {
    let map = {}
    for (let i=0;i < nums.length;i++){
        if (map[nums[i]] !== undefined){
            return [map[nums[i]], i]
        }
        map[target-nums[i]] = i
    }
};
```

TWO SUM SORTED

/*
Two Sum II - Input array is sorted
Given an array of integers that is already sorted in ascending order, find two numbers such that they add up to a specific target number.

The function twoSum should return indices of the two numbers such that they add up to the target, where index1 must be less than index2.

Note:

Your returned answers (both index1 and index2) are not zero-based.
You may assume that each input would have exactly one solution and you may not use the same element twice.
Example:

Input: numbers = [2,7,11,15], target = 9
Output: [1,2]
Explanation: The sum of 2 and 7 is 9. Therefore index1 = 1, index2 = 2.
*/

```
/**
 * @param {number[]} numbers
 * @param {number} target
 * @return {number[]}
 */
var twoSum = function(numbers, target) {
   let smallIndex = 0
   let largeIndex = numbers.length - 1

   for (let i = 0; i < numbers.length; i++){
      let sum = numbers[smallIndex] + numbers[largeIndex]
      if (sum > target){
         largeIndex--
      } else if (sum < target){
         smallIndex++
      } else {
         return [smallIndex+1, largeIndex+1]
      }
   }
};
```

URLIFY

```
// URLify == Convert spaces to %20 with true length of string given.

function URLify(str, trueLength = str.length){
   // first pass:: find the count of non space characters in the string
   //subtract the chars from the true length to see how many spaces are allowed to replace with %20

   // second pass:: if we see a space and there are still spaces left, append '%20' to output string
   // otherwise copy the current character.
   // when run out of spaces, append the empty string instead.

   let out = ''
   let trueCharacters = 0;

   for (let i = 0; i < str.length; i++){
      if (str[i] !== ' ') {
         trueCharacters++;
      }
   }

      let trueSpaces = trueLength - trueCharacters;

   for(let i = 0; i < str.length; i++){
      if (str[i] === ' ' && trueSpaces > 0){
         out = out + '%20'
         trueSpaces--;
      } else if (str[i] !== ' '){
         out = out + str[i]
      }
   }

   // if n is not yet reached there are still spaces left
   while (trueSpaces > 0){
      out = out + '%20'
      trueSpaces--;
   }
   return out;
}

console.log(
   URLify('Mr John Smith ', 13) === 'Mr%20John%20Smith',
   URLify('Mr John Smith ', 14) === 'Mr%20John%20Smith%20',
   URLify('   hi', 7) === '%20%20%20hi%20%20',
   URLify('   hi ', 3) === '%20hi',
   URLify('', 0) === '',
   URLify('', 2) === '%20%20',
   URLify('hel lo', 5) === 'hello'
 );
```

VALID PALINDROME

```
/*
Given a string, determine if it is a palindrome, considering only alphanumeric characters and
ignoring cases.

Note: For the purpose of this problem, we define empty string as valid palindrome.

Example 1:

Input: "A man, a plan, a canal: Panama"
Output: true
Example 2:

Input: "race a car"
Output: false
*/

/**
 * @param {string} s
 * @return {boolean}
 */
var isPalindrome = function(s) {
   var strippedString = s.replace(/[^\w]/g, '');
   var reversedString = strippedString.split('').reverse().join('');

   return strippedString.toLowerCase() == reversedString.toLowerCase();
};
```

VALID PARENTHESIS

```
/**
 * @param {string} s
 * @return {boolean}
 */

 // My Implementation
var isValid = function(s) {
   var stack = []
   if (s === ""){
      return true
   }
   if (s.length < 2){
      return false
   }
   for (let char of s) {
     if (char === '(' || char === '{' || char === '[') {
         stack.push(char)
      } else {
         let topElement = stack.pop();
```

```
            if (char === ")" && topElement === "("){
                continue;
            } else if (char === "}" && topElement === "{"){
                continue;
            } else if (char === "]" && topElement === "["){
                continue;
            } else {
                return false
            }
        }
    }
    if (stack.length !== 0){
        return false
    }
    return true;
};

// Proper Implementation

/**
 * @param {string} s
 * @return {boolean}
 */
var isValid = function(s) {
    var stack = []
    var parenMap = {
        '{' : '}',
        '[' : ']',
        '(' : ')',
    }

    for (let char of s){
        if (parenMap[char]){
            stack.push(parenMap[char])
        } else {
            if (char !== stack.pop()){
                return false;
            }
        }
    }

    return stack.length === 0

};
```

ZERO MATRIX

```
function zeroMatrix(m){
   let row = []
   let column = []
   for (let i = 0;i < m.length; i++){
      row.push(false);
   }
   for (let j = 0;j < m[0].length; j++){
      column.push(false);
   }
   for (let i = 0; i < m.length; i++){
      for (let j = 0; j < m[0].length; j++){
         if (m[i][j] === 0){
            row[i] = true
            column[j] = true
         }
      }
   }

   for (let i = 0;i < row.length;i++){
      if (row[i] === true){
         for (let j = 0; j < m[0].length; j++){
            m[i][j] = 0
         }
      }
   }

   for (let j = 0;j < column.length;j++){
      if (column[j] === true){
         for (let i = 0; i < m.length; i++){
            m[i][j] = 0
         }
      }
   }
   console.log(m)
   return m;
}

zeroMatrix([[0,2,3],[0,0,1],[9,7,4]])
```