**LeetCode Problem Solve**

**1.Two Sum**

Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to *target*.

You may assume that each input would have **exactly one solution**, and you may not use the same element twice.

You can return the answer in any order.

**Example 1:**

**Input:** nums = [2,7,11,15], target = 9

**Output:** [0,1]

**Explanation:** Because nums[0] + nums[1] == 9, we return [0, 1].

**Example 2:**

**Input:** nums = [3,2,4], target = 6

**Output:** [1,2]

**Constraints:**

* 2 <= nums.length <= 104
* -109 <= nums[i] <= 109
* -109 <= target <= 109
* **Only one valid answer exists.**

**Follow-up:**Can you come up with an algorithm that is less than O(n2) time complexity?

Solution:

1.

let twoSum = function(nums, target) {

  for (let i = nums.length - 1; i >= 0; i--) {

    for (let j = 0; j < i; j++) {

      if ( addition(nums[i], nums[j]) === target ) {

        return [j, i];

      }

    }

  }

};

// add a with b and return it

let addition = function(a, b) {

  return a + b;

};

2.

var twoSum = function(nums, target) {

  var hash = {};

  var len = nums.length;

  for (var i = 0; i < len; i++) {

    if (nums[i] in hash) return [hash[nums[i]], i];

    hash[target - nums[i]] = i

  }

  return [-1, -1];

};

3.

const twoSum = (nums, target) => {

  const numsObject = {};

  for(const index in nums){

      if(numsObject[target-nums[index]]){

          return [numsObject[target-nums[index]], index];

      }

      numsObject[nums[index]] = index;

  }

  return [];

};

**9. Palindrome Number**

Given an integer x, return true if x is palindrome integer.

An integer is a **palindrome** when it reads the same backward as forward.

* For example, 121 is a palindrome while 123 is not.

**Example 1:**

**Input:** x = 121

**Output:** true

**Explanation:** 121 reads as 121 from left to right and from right to left.

**Example 2:**

**Input:** x = -121

**Output:** false

**Explanation:** From left to right, it reads -121. From right to left, it becomes 121-. Therefore it is not a palindrome.

**Example 3:**

**Input:** x = 10

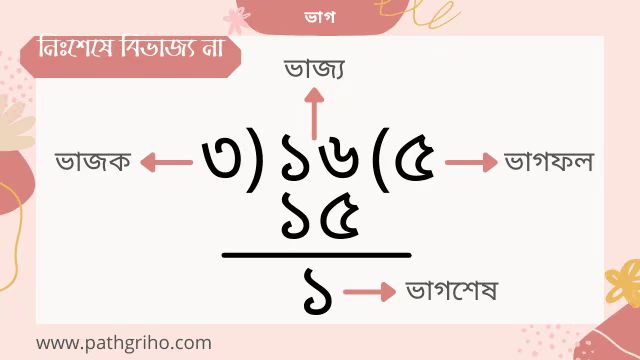
**Output:** false

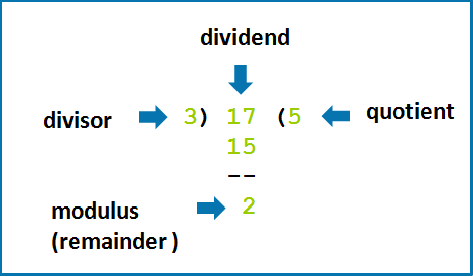
**Explanation:** Reads 01 from right to left. Therefore it is not a palindrome.

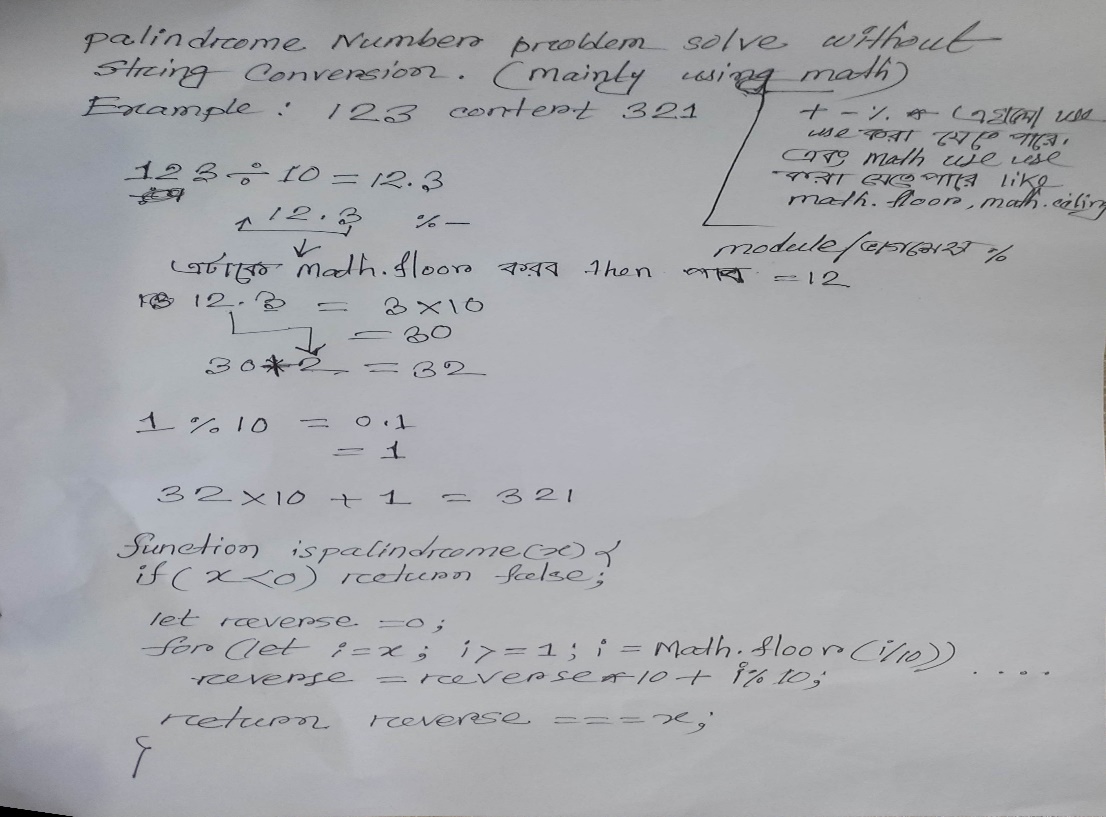
**Constraints:**

* -231 <= x <= 231 – 1

**Follow up:** Could you solve it without converting the integer to a string?







Solution:

var isPalindrome = function(x) {

if (x < 0) return false;

var num = x;

var res = 0;

while (num !== 0) {

res = (res \* 10) + (num % 10);

num = Math.floor(num / 10);

}

return res === x;

};

**Complexity:** Time complexity : O(log(n)).

Space complexity : O(1).

var isPalindrome = function(x) {

  // Base condition

  if (x < 0) {

      return false;

  }

  // Store the number in a variable

  let number = x;

  // This will store the reverse of the number

  let reverse = 0;

  while (number > 0) {

      reverse = reverse \* 10 + number % 10;

      number = parseInt(number / 10);

  }

  return x === reverse;

};

Simple Solution:

var isPalindrome = function(x) {

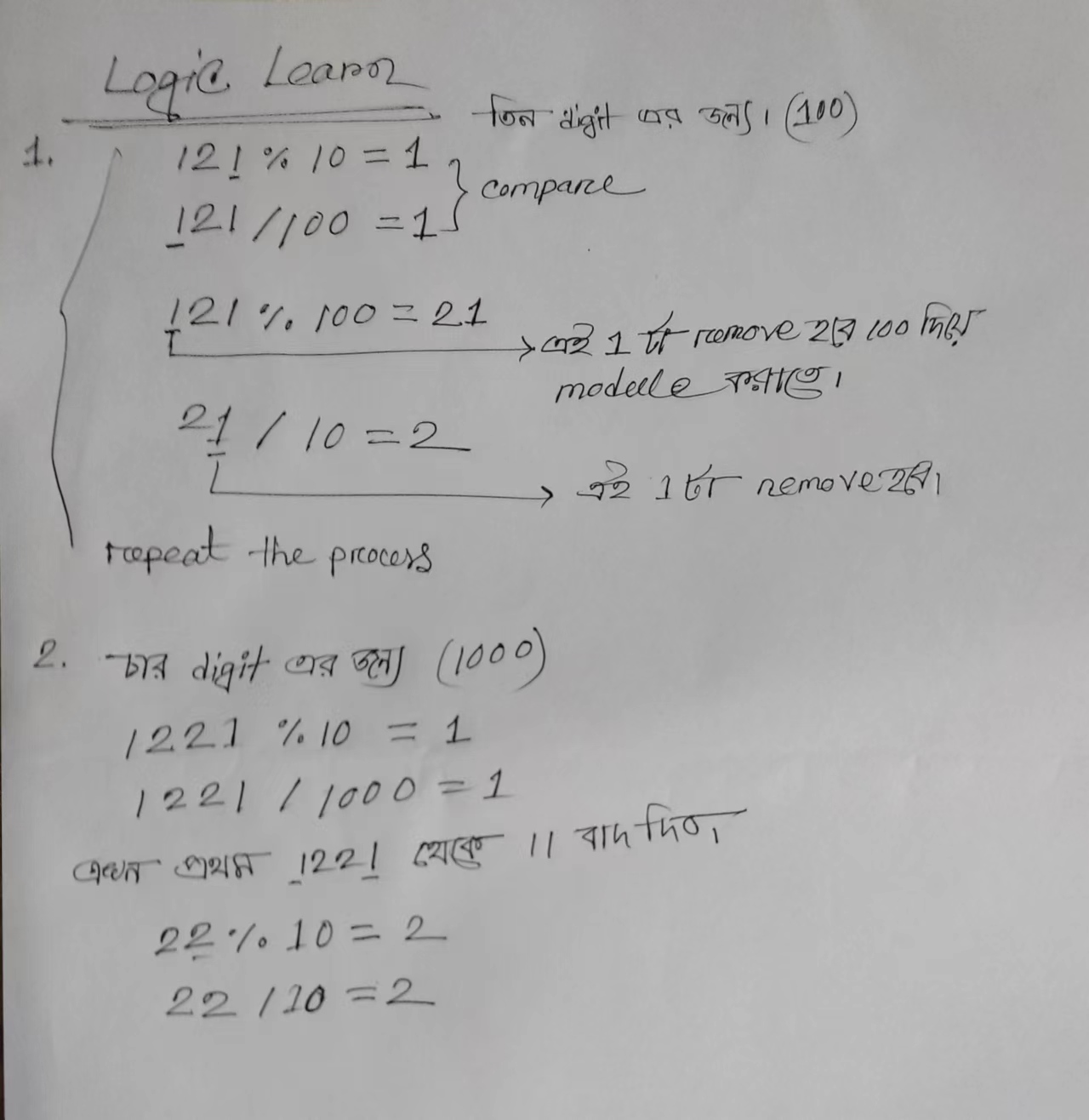
  let reversed = x.toString().split('').reverse().join('')

  return (x.toString() === reversed)

};

console.log(isPalindrome(121));

Output: true



**13. Roman to Integer**

Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M.

**Symbol** **Value**

I 1

V 5

X 10

L 50

C 100

D 500

M 1000

For example, 2 is written as II in Roman numeral, just two ones added together. 12 is written as XII, which is simply X + II. The number 27 is written as XXVII, which is XX + V + II.

Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used:

* I can be placed before V (5) and X (10) to make 4 and 9.
* X can be placed before L (50) and C (100) to make 40 and 90.
* C can be placed before D (500) and M (1000) to make 400 and 900.

Given a roman numeral, convert it to an integer.

**Example 1:**

**Input:** s = "III"

**Output:** 3

**Explanation:** III = 3.

**Example 2:**

**Input:** s = "LVIII"

**Output:** 58

**Explanation:** L = 50, V= 5, III = 3.

**Example 3:**

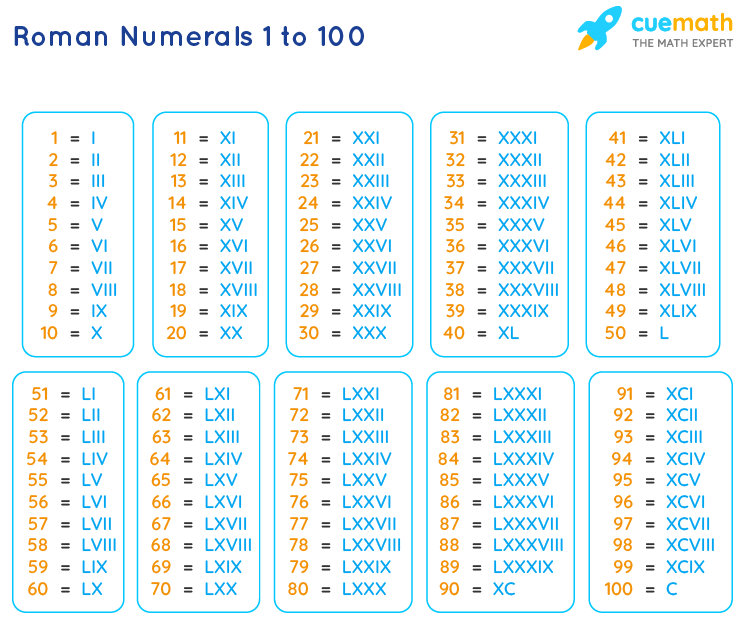
**Input:** s = "MCMXCIV"

**Output:** 1994

**Explanation:** M = 1000, CM = 900, XC = 90 and IV = 4.

**Constraints:**

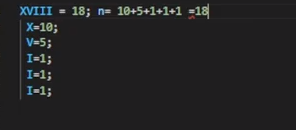
* 1 <= s.length <= 15
* s contains only the characters ('I', 'V', 'X', 'L', 'C', 'D', 'M').
* It is **guaranteed** that s is a valid roman numeral in the range [1, 3999].

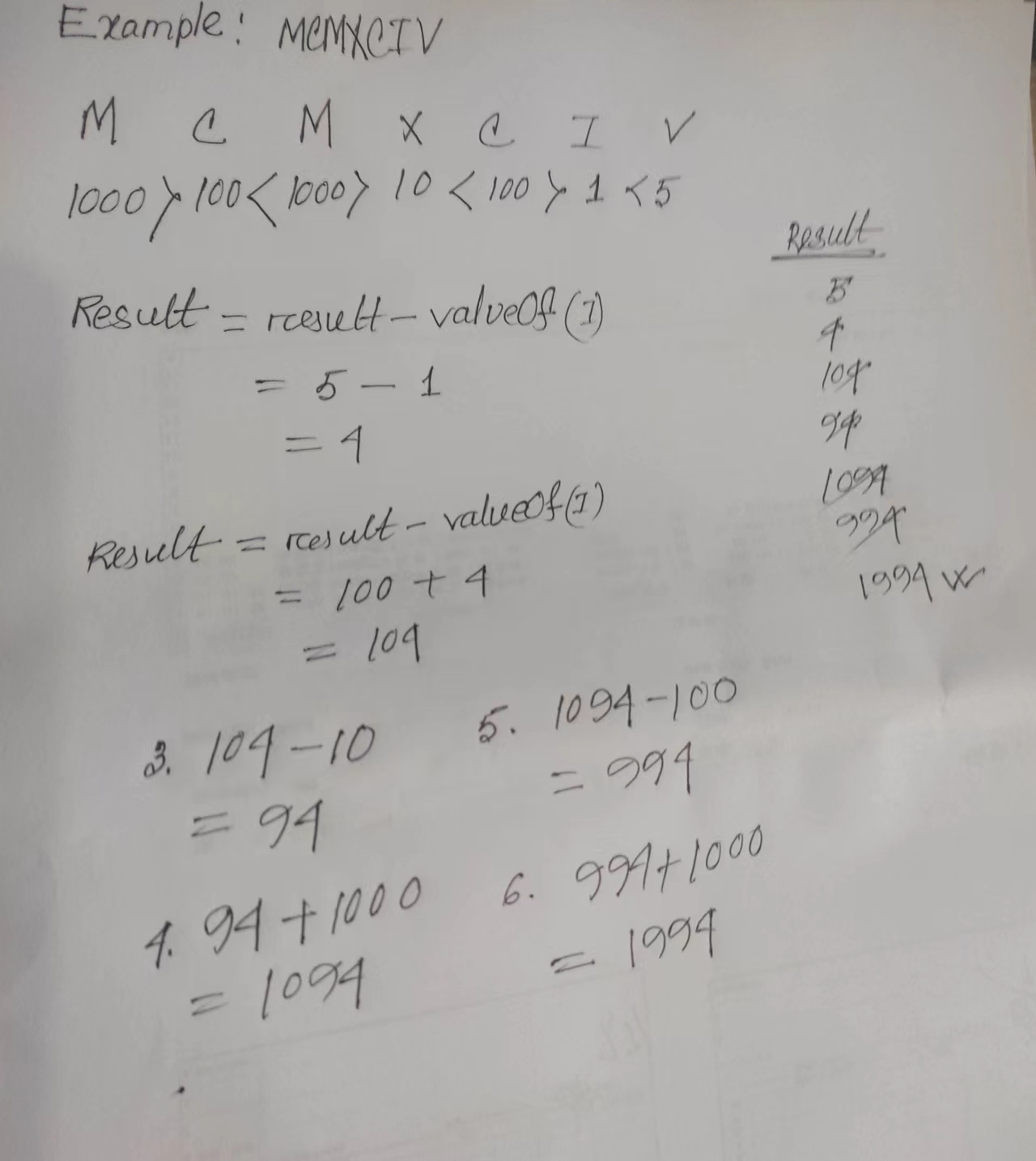


JavaScript String substr()

let text = "Hello world!";  
let result = text.substr(1, 4);

**Output**: ello





function RomantoInt(romanStr){

  let num = 0;

  let objRoman = {M:1000,D:500,C:100,L:50,X:10,V:5,I:1};

for(let i=0 ; i<romanStr.length ; i++){

      if(objRoman[romanStr[i]] < objRoman[romanStr[i+1]]){

          num -= objRoman[romanStr[i]];

      }

      else{

          num += objRoman[romanStr[i]];

      }

}

return num;

}

console.log(RomantoInt("XVIII"))

var romanToInt = function(s) {

var map = {

I: 1,

IV: 4,

V: 5,

IX: 9,

X: 10,

XL: 40,

L: 50,

XC: 90,

C: 100,

CD: 400,

D: 500,

CM: 900,

M: 1000

};

var len = s.length;

var i = 0;

var res = 0;

while (i < len) {

if (map[s.substr(i, 2)]) {

res += map[s.substr(i, 2)];

i += 2;

} else {

res += map[s[i]];

i += 1;

}

}

return res;

};

var romanToInt = function(s) {

  const sym = {

      'I': 1,

      'V': 5,

      'X': 10,

      'L': 50,

      'C': 100,

      'D': 500,

      'M': 1000

  }

  let result = 0;

  for (i=0; i < s.length; i++){

      const cur = sym[s[i]];

      const next = sym[s[i+1]];

      if (cur < next){

          result += next - cur // IV -> 5 - 1 = 4

          i++

      } else {

          result += cur

      }

  }

  return result;

};

console.log(romanToInt("IX"))