|  |
| --- |
| ### 1-reverse-string.js |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  reverseString1: (using 'array reverse')  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  [str --to--> array] --> reverse() --> [array --to-- str]  'jag' --> ['j','a','g']-->reverse() --> ['g','a','j'] --> gaj  outputStr = inputStr.split('').reverse().join('');  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  reverseString2: (using 'reverse iteration')  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  var outputStr = '';  for(var i= inputStr.length-1; i>=0; i--){  outputStr = inputStr + outputStr  }    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  reverseString2: (using 'forward iteration')  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  var outputStr = '';  for(var i= inputStr.length-1; i>=0; i--){  outputStr = outputStr + inputStr;  }    … |

|  |
| --- |
| ### 2-is-panidrome.js |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  isPalindrome1: (using 'reverseString')  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  inputStr = 'racecar'  if (inputStr === reverseString(inputStr)){    // isPalindrome  }  Note: reverseString is 'n' itertaion O(N)  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  isPalindrome2: (using 'forLoop' iterate only half the string)  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  inputStr = 'racecar'  n = inputStr.length // 7  0 1 2 3 4 5 6  r a c e c a r (7/2 = 3.5)  loop: (0 to n/2)  inputStr[0] === inputStr[6] // r  inputStr[1] === inputStr[5] // a  inputStr[2] === inputStr[4] // c  break; and return false; (in any of the above case fails)  Note: in each iteration 'indexFromLast' = lastIndex - currentIndex;   |  | | --- | | function **isPalindrome3**(str) {  var isPalindrome = true;  for(var i=0; i<Math.**round**(str.length/2); i++){  *// indexFromLast = lastIndex-currentIndex;*  var indexFromLast = (str.length-1)-i;  if(str[i] !== str[indexFromLast]){  isPalindrome = false;  break;  }  }  return isPalindrome;  } |   … |

|  |
| --- |
| ### 3-reverse-int.js |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  reverseInt1: (using 'reverseString')  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  inputNo = 1234  inputStr = inputNo.toString(); // '1234'  return parseInt(reverseString(inputStr)) // 4321  What if number is negative?  ~~~~~~~~~~~~~~~~~~~~~~~~~~~  inputNo = -1234  inputStr = inputNo.toString(); // '-1234'  isNegativeNo = Math.sign(inputNo); // -1  if (isNegativeNo === -1){  inputStr = inputStr.substring(1) ; // '1234'  }  return parseInt(reverseString(inputStr)) \* isNegativeNo; // (4321 \* -1) // -4321    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  reverseInt2: (using 'number manipulation')  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  1234 ---> 4321  0 \* 10 = 0 ===> 0 + 4 ===> 4  4 \* 10 = 40 ===> 40 + 3 ===> 43  43 \* 10 = 430 ===> 430 + 2 ===> 432  432\* 10 = 4320 ===> 4320+ 1 ===> 4321    quotient (next) reminder result(step1) result(step2) (input = quotient)  1234/10 = 123.4 ==> 1234%10 = 4 ==> 0\*10 = 0 ==> 0 + 4 = 4  123/10 = 12.3 ==> 123%10 = 3 ==> 4\*10 = 40 ==> 40 + 3 = 43  12/10 = 1.2 ==> 12%10 = 2 ==> 43\*10 = 430 ==> 430 + 2 = 432  1/10 = 0.1 ==> 1%10 = 1 ==> 432\*10 = 4320 ==> 4320 + 1 = 4321   |  | | --- | | function **reverseInt2**(input) {  var result = 0;  *// convert: negative to positive (if applicable)*  var signNo = Math.**sign**(input);  input = input \* signNo;  while (input != 0) {  var quotient = Math.**floor**(input / 10); *// 1234*  var reminder = input % 10;    *// result (step1):*  result = result \* 10;    *// result (step2):*  result = result + reminder  *// next:*  input = quotient;  }  *// convert-back: positive to negative (if applicable)*  return result \* signNo;  } |   … |

|  |
| --- |
| ### 4-find-most-repeated-char.js |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  findMostRepeatedChar('abccccccd') :  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  1. iterate: eachChar and populate 'charMap' { eachChar : its count }  -- var charMap = { a: 1, b: 1, c: 6, d: 1 }    1.1-- also find max value's 'key' in the same for loop  2. directly return the max value's 'key'     |  | | --- | | function **findMostRepeatedChar3**(str) {  var charMap = {};  var maxVal = 0;  var maxKey = '';  *// 1. iterate: eachChar and populate 'charMap' { eachChar : its count }*  for (const eachChar of str) {  if (charMap[eachChar]) {  charMap[eachChar] = charMap[eachChar] + 1;  *// 1.1-- also find max value's 'key' in the same for loop*  if (charMap[eachChar] > maxVal) {  maxVal = charMap[eachChar];  maxKey = eachChar;  }  } else {  charMap[eachChar] = 1  }  }  **console**.**log**(charMap);  return maxKey;  } |   … |

|  |
| --- |
| ### 5-fizz-buzz.js |
| Write a program to print numbr from 1 to n.  -- But for multiples of three print “fizz” instead of the number  -- and for the multiples of five print “buzz”.  -- For numbers which are multiples of both three and five print “fizzbuzz”.  Understanding:  --------------  multiples of number means reminder is zero (Use 'modulo' operator % to get reminder)  3/3 ==> Q = 1 , 3%3 ==> Reminder = 0  9/3 ==> Q = 3 , 9%3 ==> Reminder = 0  12/3 ==> Q = 4 , 12%3 ==> Reminder = 0  Solution:  ----------  i: 1 ==> (i % 3 = 1) or (i % 5 = 1) ==> 1  i: 2 ==> (i % 3 = 2) or (i % 5 = 2) ==> 2  i: 3 ==> (i % 3 = 0) or (i % 5 = 3) ==> fizz  i: 4 ==> (i % 3 = 1) or (i % 5 = 4) ==> 4  i: 5 ==> (i % 3 = 2) or (i % 5 = 0) ==> buzz  i: 6 ==> (i % 3 = 0) or (i % 5 = 1) ==> fizz  i: 7 ==> (i % 3 = 1) or (i % 5 = 2) ==> 7  i: 8 ==> (i % 3 = 2) or (i % 5 = 3) ==> 8  i: 9 ==> (i % 3 = 0) or (i % 5 = 4) ==> fizz  i: 10 ==> (i % 3 = 1) or (i % 5 = 0) ==> buzz  i: 11 ==> (i % 3 = 2) or (i % 5 = 1) ==> 11  i: 12 ==> (i % 3 = 0) or (i % 5 = 2) ==> fizz  i: 13 ==> (i % 3 = 1) or (i % 5 = 3) ==> 13  i: 14 ==> (i % 3 = 2) or (i % 5 = 4) ==> 14  i: 15 ==> (i % 3 = 0) and (i % 5 = 0) ==> fizzbuzz   |  | | --- | | function **fizzBuzz**(n) {  *// loop: (1 to n)*  for (var i = 1; i <= n; i++) {  *// default: is numbr*  var result = i;    if (i % 3 === 0 && i % 5 === 0) {  *//* *current numbr is multiple of 3 & 5*  result = 'fizzbuzz';  } else if (i % 3 === 0) {  *// current numbr is multiple of 3*  result = 'fizz';  } else if (i % 5 === 0) {  *// current numbr is multiple of 5*  result = 'buzz';  }  **console**.**log**(result);  }  } |   … |

|  |
| --- |
| ### 6-chunk-array.js |
| Given an array and chunk maxSubArrSize,  --divide the array into many subarrays where each subarray is of length maxSubArrSize  // chunk([1, 2, 3, 4], 2) --> [[ 1, 2], [3, 4]]  // chunk([1, 2, 3, 4, 5], 2) --> [[ 1, 2], [3, 4], [5]]  // chunk([1, 2, 3, 4, 5, 6, 7, 8], 3) --> [[ 1, 2, 3], [4, 5, 6], [7, 8]]  // chunk([1, 2, 3, 4, 5], 4) --> [[ 1, 2, 3, 4], [5]]  // chunk([1, 2, 3, 4, 5], 10) --> [[ 1, 2, 3, 4, 5]]  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  chunkArray1 : (by getting 'lastSubArr' from 'resultArr')  -if space exists: add item in 'lastSubArr'  -or : add new 'subArr' into 'resultArr'  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  init: resultArr = []  // loop: (0 to arr length)  {  1. get: last SubArr in resultArr  2. check: 'lastSubArr' exists and 'lastSubArr' has enough space to add 'new' item  --then: // add new item to 'lastSubArr'  --else:  `case1: lastSubArr not available`  `-or-`  `case2: lastSubArr doesnt have enough space to add new item`  // create new 'subArr' and add to resultArr  }     |  | | --- | | function **chunkArray1**(inputArr, maxSubArrSize) {  var resultArr = [];  *// loop: (0 to arr length)*  for (var i = 0; i < inputArr.length; i++) {  *// 1. get: last SubArr in resultArr*  var lastSubArr = resultArr[resultArr.length - 1];  *// 2. check: 'lastSubArr' exists and 'lastSubArr' has enough space to add 'new' item*  if (lastSubArr && lastSubArr.length < maxSubArrSize) {  *// add new item to 'lastSubArr'*  lastSubArr.**push**(inputArr[i]);  } else {  *// case1: lastSubArr not available*  *// -or-*  *// case2: lastSubArr doesnt have enough space to add new item*  *// create new subArr and add to resultArr*  resultArr.**push**([inputArr[i]]);  }  }  return resultArr;  } | |  |   \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  chunkArray2 : (using arr 'slice')  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  arr.slice(startIndex, endIndex);  chunkArray2([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11], 3)    endIndex(startIndex + maxSubArrSize)  ---------------------  startIndex: 0, endIndex (0 + 3): 3 ==> sliced arr : 1,2,3  startIndex: 3, endIndex (3 + 3): 6 ==> sliced arr : 4,5,6  startIndex: 6, endIndex (6 + 3): 9 ==> sliced arr : 7,8,9  startIndex: 9, endIndex (9 + 3): 12 ==> sliced arr : 10,11    loop: (until index is less than arr length)  // (means: when index is beyond the size of the arr, break; it)  {  // 1. slice: the Array (startIndex & endIndex)  --endIndex: currentIndex + maxSubArrSize  // 2. add: the sliced array  // 3. next: ('endIndex' --is--> next iteration's 'startIndex')  }  ----in every end of iteration, ('endIndex' --is--> next iteration's 'startIndex')     |  | | --- | | function **chunkArray2**(inputArr, maxSubArrSize) {  var resultArr = [];  var currentIndex = 0;  *// loop: (until 'currentIndex' is less than inputArr length)*  *// (means: when 'currentIndex' is beyond the size of the inputArr, break; it)*  while (currentIndex < inputArr.length) {  *// 1. slice: the Array*  var startIndex = currentIndex;  var endIndex = currentIndex + maxSubArrSize;  var subArr = inputArr.**slice**(startIndex, endIndex);  *// 2. add: the sliced array*  resultArr.**push**(subArr);  *// console.log(`startIndex: ${startIndex}, endIndex (${currentIndex} + ${maxSubArrSize}): ${endIndex} ==> sliced arr : ${subArr}`);*  *// 3. next: ('endIndex' --is--> next iteration's 'startIndex')*  currentIndex = endIndex;  }  return resultArr;  } |   … |

|  |
| --- |
| ### 7-is-anagram.js |
| isAnagram: (check two provided strings are Anagram)  -it is 'anagram' if both string has same characters with same quantity  -ignore: special characters  -consider: caps letter as 'small case'  Examples:  ----------  isAnagram1('rail safety', 'fairy tales')) //true  isAnagram1('RAIL! SAFETY!', 'fairy tales')) //true  isAnagram1('Hi there', 'Bye there')) //false  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  isAnagram1 : (get 'charMap' for both string, 'iterate' them and identify)  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  isAnagram1('hello', 'llohe')  ## Step0: Clean String  var cleanedStr = stringA.replace(/[^A-Za-z0-9]/g, '').toLowerCase();  ## Step1: Populate charMap  'hello' --> Character Map  {  h: 1,  e: 1,  l: 2,  o: 1  }  'llohe' --> Character Map  {  l: 2,  o: 1,  h: 1,  e: 1  }  ## Step2: Check both 'charMap' keys length (should be same)  ## Step3: Check both 'charMap' has same value for the same key   |  | | --- | | function **\_getCharMap**(inputStr) {  var charMap = {};  for (let i = 0; i < inputStr.length; i++) {  if (charMap[inputStr[i]]) {  charMap[inputStr[i]] = charMap[inputStr[i]] + 1;  } else {  charMap[inputStr[i]] = 1;  }  *// -or- (1 liner)*  *// charMap[inputStr[i]] = charMap[inputStr[i]]+1 || 1;*  }  return charMap;  }  function **isAnagram1**(stringA, stringB) {  var isAnagram = true;  *// clean: string (remove unwanted charcters)*  var aCleanStr = stringA.**replace**(/[^A-Za-z0-9]/g, '').**toLowerCase**();  var bCleanStr = stringB.**replace**(/[^A-Za-z0-9]/g, '').**toLowerCase**();  *// 1: Populate charMap*  var aCharMap = **\_getCharMap**(aCleanStr);  var bCharMap = **\_getCharMap**(bCleanStr);  var aKeys = **Object**.**keys**(aCharMap);  var bKeys = **Object**.**keys**(bCharMap);  *// 2: both has same no. of keys*  if (aKeys.length === bKeys.length) {  for (let key in aCharMap) {  var aVal = aCharMap[key];  var bVal = bCharMap[key];  if (aVal !== bVal) {  *// both charMap does NOT have same value for the same key*  isAnagram = false;  break;  }  *// 3: both 'charMap' has same value for the same key*  }  } else {  isAnagram = false;  }  return isAnagram;  } |   \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  isAnagram2 : (transform to 'array', 'sort' them and identify)  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  'sort': is performance impact  'hello' ---str.split('')---> ['h', 'e', 'l', 'l', 'o'] ---arr.sort()---> ['e', 'h', 'l', 'l', 'o'] ---arr.join('')---> 'ehllo'  'llohe' ---str.split('')---> ['l', 'l', 'o', 'h', 'e'] ---arr.sort()---> ['e', 'h', 'l', 'l', 'o'] ---arr.join('')---> 'ehllo'  'ehllo' === 'ehllo' // is anagram :)   |  | | --- | | function **cleanSortJoinStr**(str) {  return str.**replace**(/[^A-Za-z0-9]/g, '').**toLowerCase**() *//clean*  .**split**('') *// str to arr*  .**sort**() *// sort // Remember: performance overhead*  .**join**(''); *// arr to str*  }  function **isAnagram3**(stringA, stringB) {  return **cleanSortJoinStr**(stringA) === **cleanSortJoinStr**(stringB);  } |   … |

|  |
| --- |
| ### 8-capitalize-each-word.js |
| capitalize the first letter of each word in the given string  Examples:  ----------  capitalize('a short story') // 'A Short Story'  capitalize('a lazy fox') // 'A Lazy Fox'  capitalize('look, it is working!') // 'Look, It Is Working!'  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  capitalize1 : (transform to 'array', array firstItem.toUpperCase(), and transform to 'string'  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  'a short story' --str.split(' ')--> ["a", "short", "story"] -- eachWord[0].toUpperCase() + eachWord.slice(1) --> ["A", "Short", "Story"] --str.join(' ')--> 'A Short Story'  'a lazy fox' --str.split(' ')--> ["a", "lazy", "fox"] -- eachWord[0].toUpperCase() + eachWord.slice(1) --> ["A", "Lazy", "Fox"] --str.join(' ')--> 'A Lazy Fox'   |  | | --- | | function **capitalize1**(inputStr) {  var inputWordsArr = inputStr.**split**(' ');    var outputWordsArr = [];  for (let eachWord of inputWordsArr) {  let upperFirstChar = eachWord[0].**toUpperCase**();  let upperCasedWord = upperFirstChar + eachWord.**slice**(1);  outputWordsArr.**push**(upperCasedWord);  }  return outputWordsArr.**join**(' ');  } |     \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  capitalize2 : (iterate: each char, if previousChar is 'space' -- capitalize the current charChar  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // 1: first char is always caps  // 2. iterate: each char (1 to str length)  // 3. Is previousChar is 'space'? if (inputStr[i-1] === ' ') { ... }  then:  // previousChar is 'space' – capitalize currentChar  // (capitalize: 'currentChar') and (append: currentChar with 'resultStr')  else:  // previousChar is NOT 'space' – do not capitalize currentChar  // (append: 'currentChar' with 'resultStr')   |  | | --- | | function **capitalize2**(inputStr) {  *// 1: first char always caps*  let resultStr = inputStr[0].**toUpperCase**();  *// 2. iterate: (1 to str length)*  for(let i=1; i< inputStr.length; i++) {  let currentChar = inputStr[i];  let previousChar = inputStr[i-1];  *// 3. Is previous char is 'space'?*  if (previousChar === ' ') {  *// previousChar is 'space' – capitalize currentChar*  *// (capitalize: current char) and (append: current char with 'resultStr')*  resultStr = resultStr + currentChar.**toUpperCase**();  } else {  *// previousChar is NOT 'space' – do not capitalize currentChar*  *// (append: current char with 'resultStr')*  resultStr = resultStr + currentChar;  }  }  return resultStr;  } |     … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |