**Student:** Yerbolat Pazyl

**Group:** IS-1808K

**ID:** 26244

Laboratory work #1. Arrays and String.

Solve all of problems for Arrays and for Strings.

Make a report on each tasks you have solved.

Deadline: 15th September 2020.

Arrays

1. <https://leetcode.com/problems/range-sum-query-immutable/>

**Description**

**303. Range Sum Query – Immutable**

Given an integer array nums, find the sum of the elements between indices i and j (i ≤ j), inclusive.

**Example:**

Given nums = [-2, 0, 3, -5, 2, -1]

sumRange(0, 2) -> 1

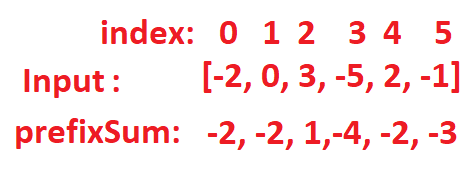
sumRange(2, 5) -> -1

sumRange(0, 5) -> -3

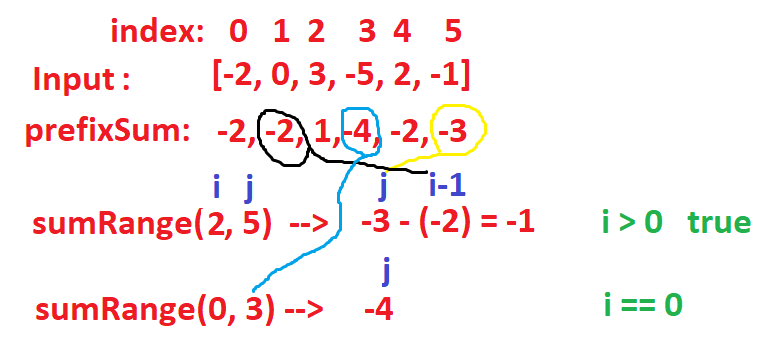
**Solution:**

First you need to find the sum of the elements in each index using the function “NumArray”.

For example: input [-2, 0, 3, -5, 2, -1], prefixSum will be [-2, -2, 1, -4, -2, -3]



Next step: if the index “i” is greater than 0 then take the sum of the prefix with index “j” from the array prefixSum and and subtract the sum of the prefix with index “(i-1)” from it. If the index “i” is 0 then to take the sum of the prefix with the index “j”.



**Java Code:**

class NumArray {

private int prefixSum [];

public NumArray(int[] nums) {

prefixSum = new int [nums.length];

for(int i = 0; i < nums.length; i++){

prefixSum[i] = nums[i];

if (i > 0)

prefixSum[i] += prefixSum[i - 1];

}

}

public int sumRange(int i, int j) {

int sum = prefixSum[j];

if (i > 0)

sum -= prefixSum[i - 1];

return sum;

}

}

1. <https://leetcode.com/problems/maximum-subarray/>

**Description**

**53. Maximum Subarray**

Given an integer array nums, find the contiguous subarray (containing at least one number) which has the largest sum and return its sum.

**Follow up:** If you have figured out the O(n) solution, try coding another solution using the **divide and conquer** approach, which is more subtle.

**Example 1:**

**Input:** nums = [-2,1,-3,4,-1,2,1,-5,4]

**Output:** 6

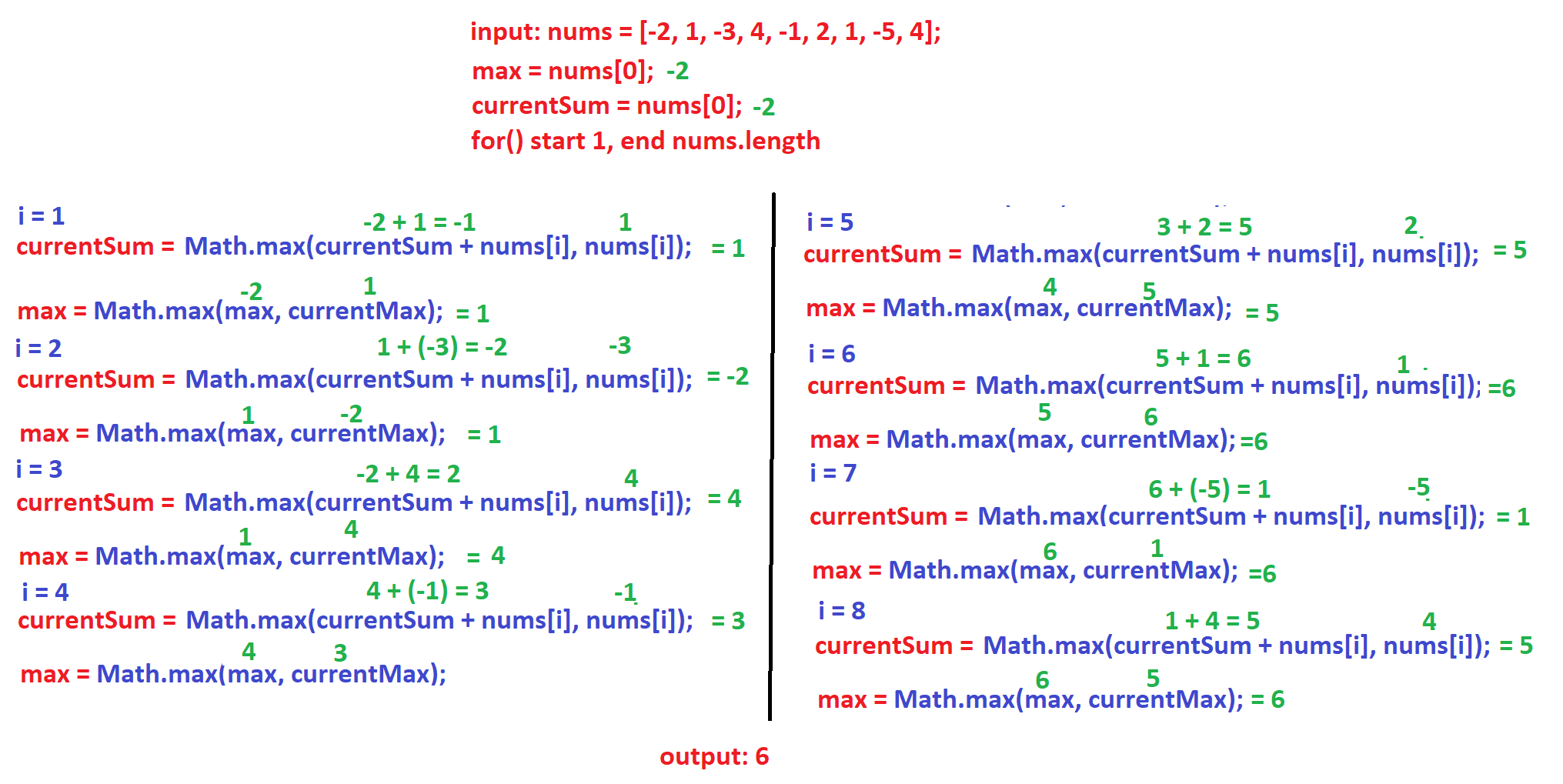
**Explanation:** [4,-1,2,1] has the largest sum = 6.

**Solution:**

To find the largest sum, we will run through each element of the array. we need two variables one for storing the largest sum(max) and the other for the current sum(currentMax).

Temporarily storing data from an array “nums” with index “0” (nums[0]) in “max” and “currentSum” variables.

Starting the loop from an index greater than 0 to the size of the array. Each time we add the nums[i] to currentMax and compare it with nums[i] (Math.max(currentMax + nums[i], nums[i])). We take the largest of these and assign it to currentMax. Finally, we compare max with currentMax and the max value will be larger of these (Math.max(currentMax, max)).



**Java Code**

class Solution {

public int maxSubArray(int[] nums) {

int maxSum = nums[0];

int currentSum = nums[0];

for(int i = 1; i < nums.length; i++) {

currentSum = Math.max(currentSum + nums[i], nums[i]);

maxSum = Math.max(currentSum, maxSum);

}

return maxSum;

}

}

1. <https://leetcode.com/problems/product-of-array-except-self/>

**Description**

**238. Product of Array Except Self**

Given an array nums of n integers where n > 1, return an array output such that output[i] is equal to the product of all the elements of nums except nums[i].

**Example:**

**Input:** [1,2,3,4]

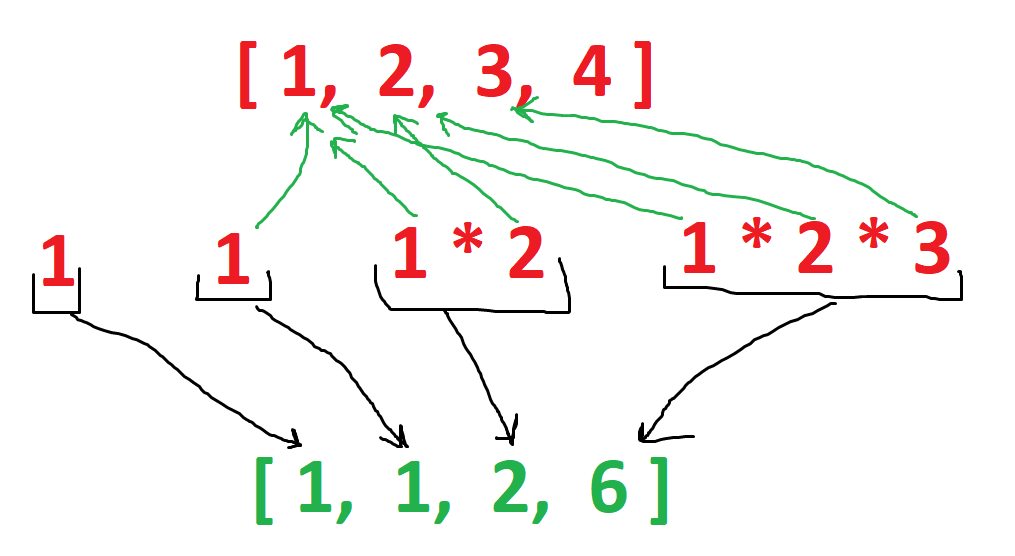
**Output:** [24,12,8,6]

**Solution:**

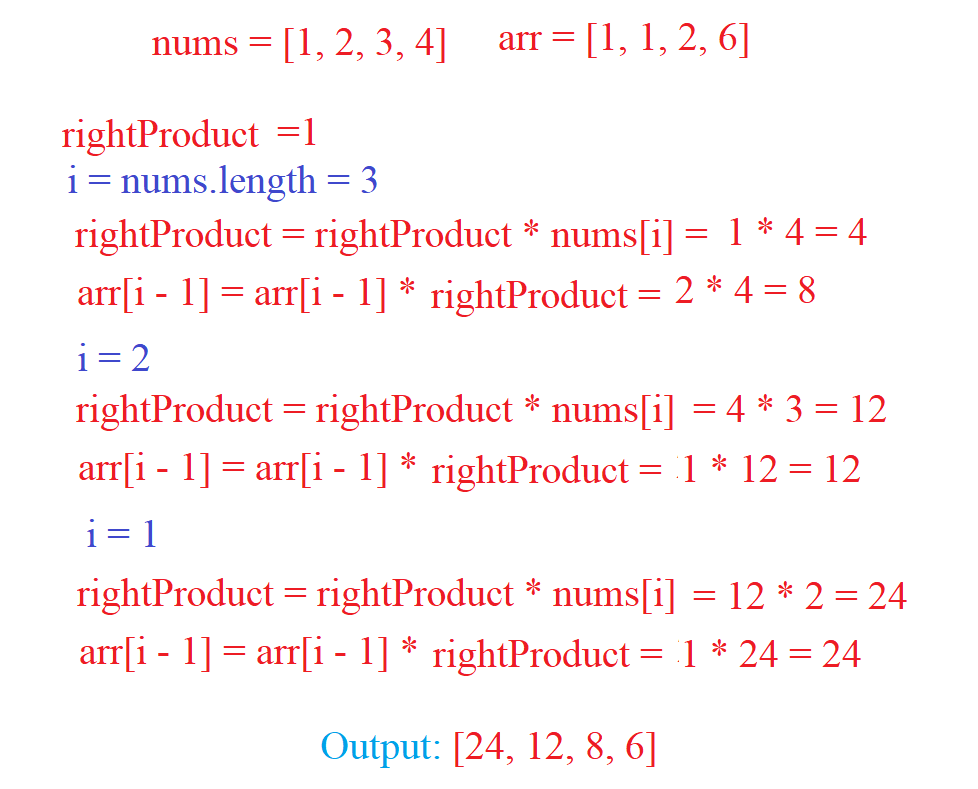
I divided this task into 2 parts.

In the first part we find the sum of multiplication from left to right. The loop starts from 1 to nums.length. Create new array “arr” with size nums.length. In “arr” array with index 0 put arr[0] = 1. Each time in the loop do arr[i] = arr[i - 1] \* nums[i - 1].

For example, if we take the array [1, 2, 3, 4], the new array will be [1, 1, 3, 6].



In the second part we create a variable rightProduct to store the sum of multiplication right to left(rightProduct = 1). The loop starts from nums.length to 0. Each time in the loop do rightProduct \*= nums[i] and arr[i - 1] \*= rightProduct.



**Java Code**

class Solution {

public int[] productExceptSelf(int[] nums) {

int n = nums.length;

int arr[] = new int[nums.length];

arr[0] = 1;

for(int i = 1; i < n; i++){

arr[i] = arr[i - 1] \* nums[i - 1];

}

int rightProduct = 1;

for(int i = n - 1; i > 0; i--) {

rightProduct \*= nums[i];

arr[i - 1] \*= rightProduct;

}

return arr;

}

}

1. <https://leetcode.com/problems/missing-number/>

**Description**

**268. Missing Number**

Given an array containing n distinct numbers taken from 0, 1, 2, ..., n, find the one that is missing from the array.

**Example 1:**

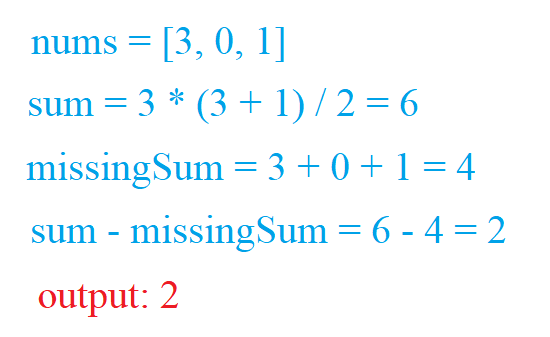
**Input:** [3,0,1]

**Output:** 2

**Solution:**

The elements in the array are not repeated and we can say that if we find the sum from 1 to n then subtract the sum of the array elements from nums, we will find missing number.

To find the sum of numbers from 1 to n there is a formula sum = n \* (n + 1) / 2.



**Java Code**

class Solution {

public int missingNumber(int[] nums) {

int sum = nums.length \* (nums.length + 1) / 2, missingSum = 0;

for(int i : nums) {

missingSum += i;

}

return sum - missingSum;

}

}

1. <https://leetcode.com/problems/maximum-average-subarray-i/>

**Description**

**643. Maximum Average Subarray I**

Given an array consisting of n integers, find the contiguous subarray of given length k that has the maximum average value. And you need to output the maximum average value.

**Example 1:**

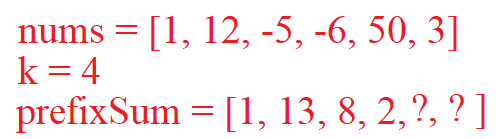
**Input:** [1,12,-5,-6,50,3], k = 4

**Output:** 12.75

**Explanation:** Maximum average is (12-5-6+50)/4 = 51/4 = 12.75

**Solution:**

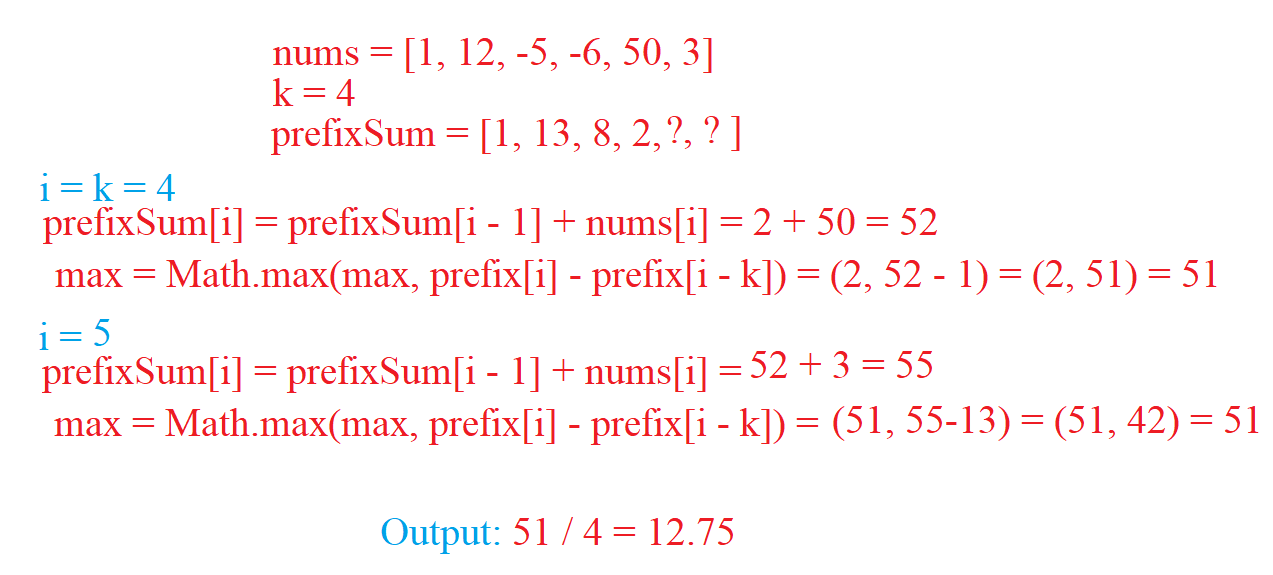
First, we find the prefix of the nums array and store it in a new array, but before the k index.



Create the max variable and put the data from the prefixSum array with the index (k-1) max=prefixSum[k-1].

Then we use a loop to run the prefix of the array with index k to nums.length. Each time we find the prefix of prefix[i] = prefix[i-1] + nums[i] and take the largest value from max and prefix[i] - prefix[i - k] then overwriting the max value.

Last max = max / k;



class Solution {

public double findMaxAverage(int[] nums, int k) {

double prefix[] = new double [nums.length];

prefix[0] = nums[0];

for(int i = 1; i < k; i++)

prefix[i] = prefix[i - 1] + nums[i];

double max = prefix[k - 1];

for(int i = k; i < nums.length; i++){

prefix[i] = prefix[i - 1] + nums[i];

max = Math.max(max, prefix[i] - prefix[i - k]);

}

return max / k;

}

}

1. <https://leetcode.com/problems/range-sum-query-2d-immutable/>

**Description**

**304. Range Sum Query 2D – Immutable**

Given a 2D matrix *matrix*, find the sum of the elements inside the rectangle defined by its upper left corner (*row*1, *col*1) and lower right corner (*row*2, *col*2).



The above rectangle (with the red border) is defined by (row1, col1) = **(2, 1)** and (row2, col2) = **(4, 3)**, which contains sum = **8**.

**Example:**

Given matrix = [

[3, 0, 1, 4, 2],

[5, 6, 3, 2, 1],

[1, 2, 0, 1, 5],

[4, 1, 0, 1, 7],

[1, 0, 3, 0, 5]

]

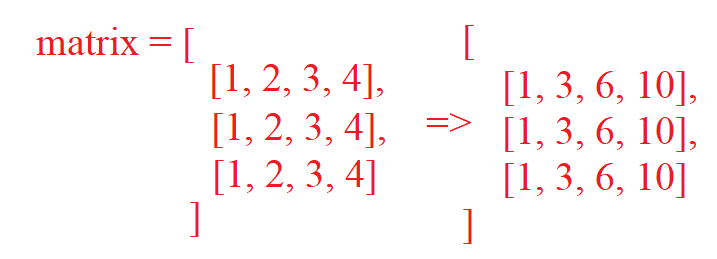
sumRegion(2, 1, 4, 3) -> 8

sumRegion(1, 1, 2, 2) -> 11

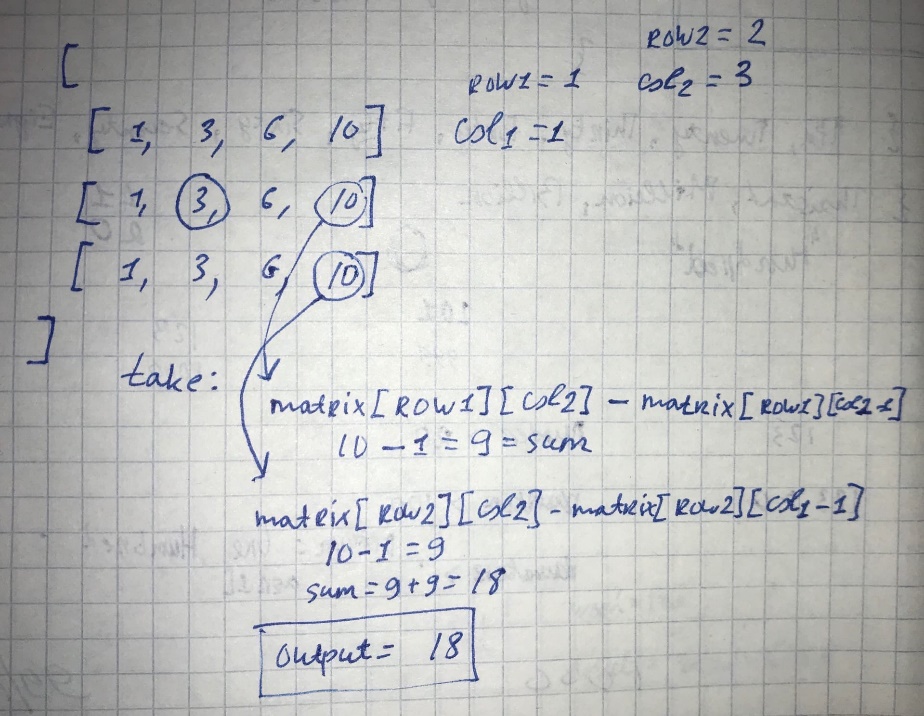
sumRegion(1, 2, 2, 4) -> 12

**Solution**

Inside the NumMatrix() function we find the prefix of each row of the 2D array, but first we check the array for emptiness.



Inside the sumRegion() function, create a variable sum with the value 0. Running the loop from index row1 to row2, each time adding the value matrix[i][col2] and if col1 > 0 subtracting from sum matrix[i][j-1]. After look return sum.



**Java Code**

class NumMatrix {

private int matrix[][];

public NumMatrix(int[][] matrix) {

if (matrix.length == 0 || matrix[0].length == 0) return;

this.matrix = new int[matrix.length][matrix[0].length];

for (int i = 0; i < matrix.length; i++){

for (int j = 0; j < matrix[i].length; j++){

this.matrix[i][j] = matrix[i][j];

if(j > 0)

this.matrix[i][j] += this.matrix[i][j - 1];

}

}

}

public int sumRegion(int row1, int col1, int row2, int col2) {

int sum = 0;

for (int i = row1; i <= row2; i++) {

sum += matrix[i][col2];

if (col1 > 0)

sum -= matrix[i][col1 - 1];

}

return sum;

}

}

1. <https://leetcode.com/problems/rotate-image/>

**Description**

**48. Rotate Image**

You are given an n x n 2D matrix representing an image, rotate the image by 90 degrees (clockwise).

You have to rotate the image in-place, which means you have to modify the input 2D matrix directly. DO NOT allocate another 2D matrix and do the rotation.

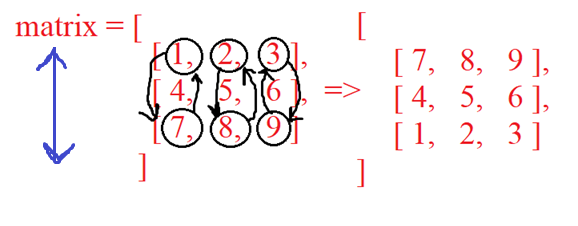


**Input:** matrix = [[1,2,3],[4,5,6],[7,8,9]]

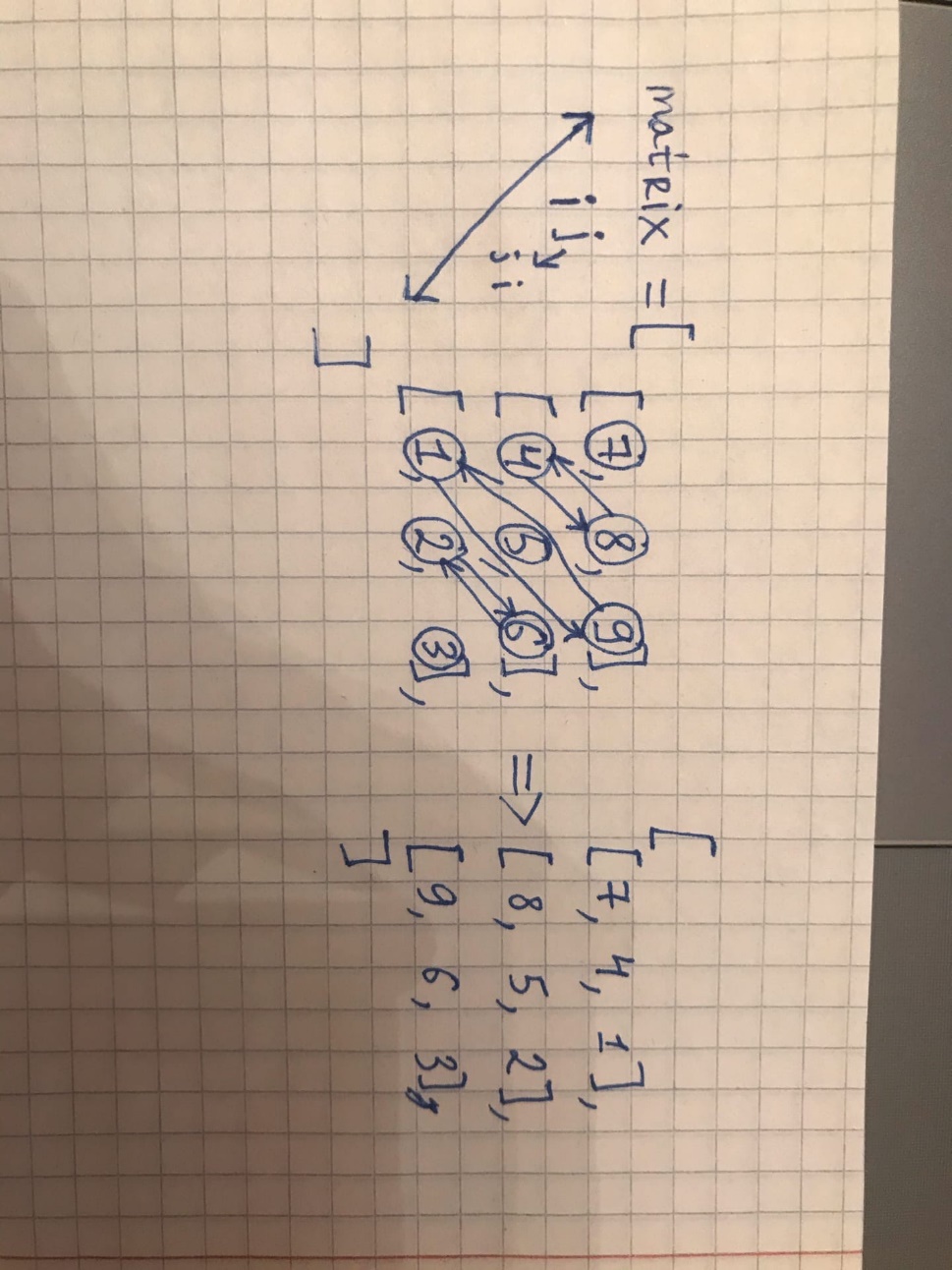
**Output:** [[7,4,1],[8,5,2],[9,6,3]]

**Solution**

First, we swap the elements vertically. For example index with value matrix[i][j] will swap with matrix[n – i – 1][j] (n is size of aray).



After vertical swap need main diagonal swap. For example, index with matrix[i][j] swap with matrix[j][i].



**Java Code**

class Solution {

public void rotate(int[][] matrix) {

int n = matrix.length;

for(int i = 0; i < n / 2; i++) {

for(int j = 0; j < n; j++) {

int temp = matrix[i][j];

matrix[i][j] = matrix[n - i - 1][j];

matrix[n - i - 1][j] = temp;

}

}

for(int i = 0; i < n; i++) {

for(int j = i; j < n ; j++) {

int temp = matrix[i][j];

matrix[i][j] = matrix[j][i];

matrix[j][i] = temp;

}

}

}

}

Strings

1. <https://leetcode.com/problems/reverse-words-in-a-string/>

**Description**

**151. Reverse Words in a String**

Given an input string, reverse the string word by word.

**Example 1:**

**Input:** "the sky is blue"

**Output:**"blue is sky the"

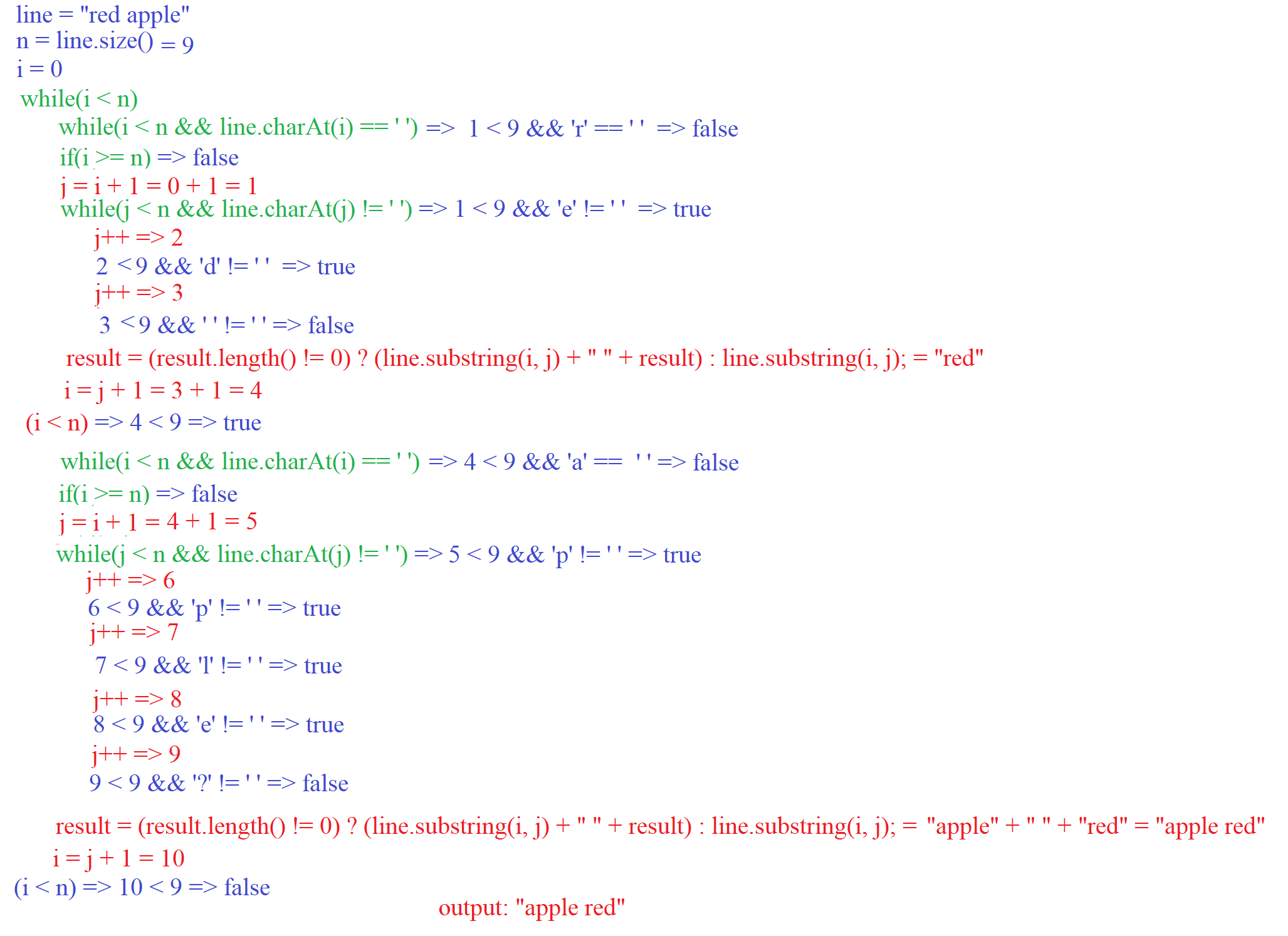
**Solution**

Сreating n, i, j, variables. The value of n is equal to the size of line, and i is 0.

Create while loop, and once again open the loop with line.charAt(i) == ‘’. In inner loop, the “i” will increase by 1 until it finds a non empty character. When the inner loop finds that it is not empty it exits the loop and checks if “i” has reached the end of the line.

If not then j will be j = i + 1. Opening the loop for j, it will increase by 1 until it finds an empty character and reaches the end of the line. When j finds an empty string it exits the loop.

If result is not empty then add (line.substring(i, j) + " " + result) or if it is empty add line.substring(i, j). Repeat all this until we reach the end of the line.

****

**Java Code**

class Solution {

public String reverseWords(String line) {

int n = line.length();

int i = 0, j = 0;

String result = new String();

while(i < n) {

while(i < n && line.charAt(i) == ' '){

i++;

}

if(i >= n)

break;

j = i + 1;

while(j < n && line.charAt(j) != ' ') {

j++;

}

result = (result.length() != 0) ? (line.substring(i, j) + " " + result) : line.substring(i, j);

i = j + 1;

}

return result;

}

}

1. <https://leetcode.com/problems/license-key-formatting/>

**Description**

**482. License Key Formatting**

You are given a license key represented as a string S which consists only alphanumeric character and dashes. The string is separated into N+1 groups by N dashes.

Given a number K, we would want to reformat the strings such that each group contains exactly K characters, except for the first group which could be shorter than K, but still must contain at least one character. Furthermore, there must be a dash inserted between two groups and all lowercase letters should be converted to uppercase.

Given a non-empty string S and a number K, format the string according to the rules described above.

**Example 1:**

**Input:** S = "5F3Z-2e-9-w", K = 4

**Output:** "5F3Z-2E9W"

**Explanation:** The string S has been split into two parts, each part has 4 characters.

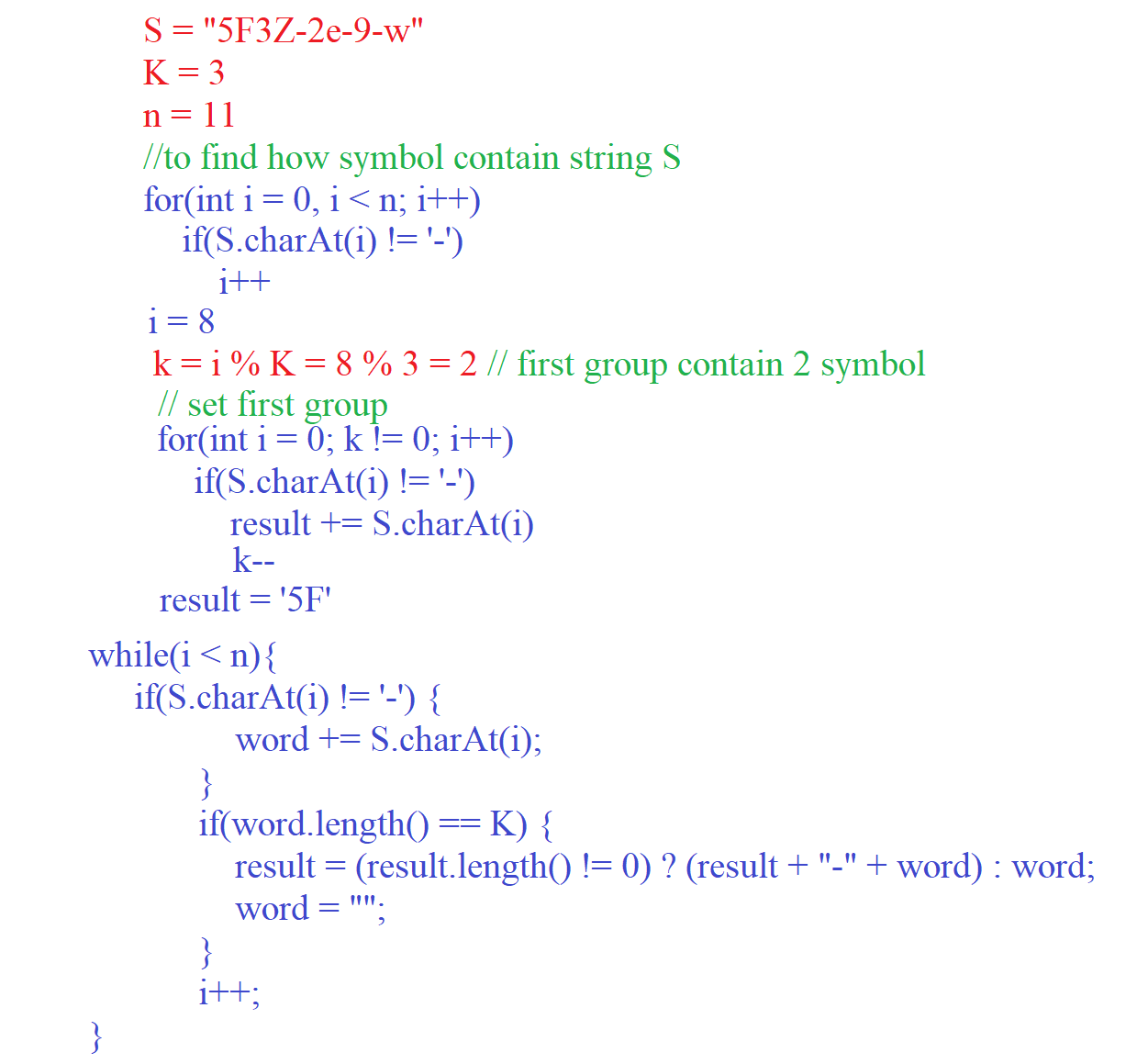
Note that the two extra dashes are not needed and can be removed.

**Solution**

Firstly, use the loop to find out how many letters S string contains, save this to the variable i. Then in the variable k we write value from (i % K), this number shows how many characters the first group will contain.

Setting the first group of characters, running through the for loop until k becomes 0, otherwise we add S.charAt(i).

Open the while loop with the condition i < n, continue adding word += S.charAt(i) until i becomes n and if word.length() becomes exactly K that do result = (result.length() != 0) ? (result + "-" + word) : word and word = "";



**Java Code**

class Solution {

public String licenseKeyFormatting(String S, int K) {

int n = S.length();

int i = 0;

String word = new String();

String result = new String();

for(int q = 0; q < n; q++) {

if(S.charAt(q) != '-')

i++;

}

int k = i % K;

for(i = 0; k != 0; i++) {

if(S.charAt(i) != '-') {

result += S.charAt(i);

k--;

}

}

while(i < n) {

if(S.charAt(i) != '-') {

word += S.charAt(i);

}

if(word.length() == K) {

result = (result.length() != 0) ? (result + "-" + word) : word;

word = "";

}

i++;

}

return result.toUpperCase();

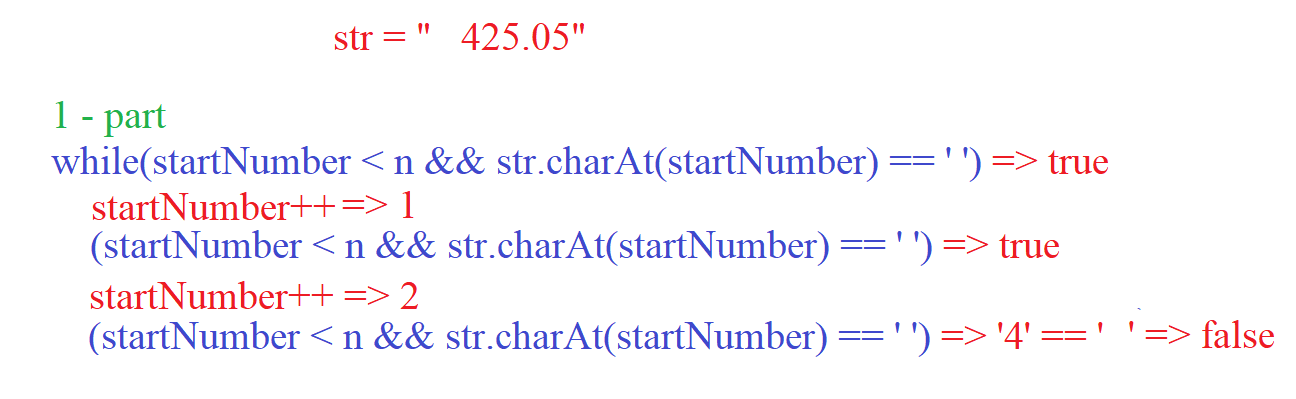
}

}

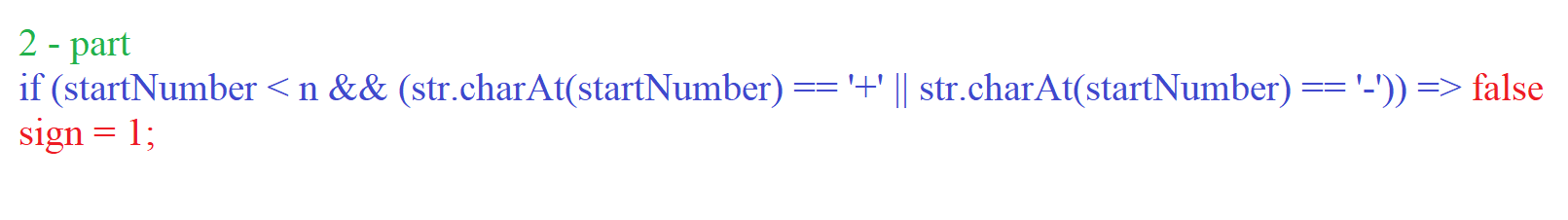
1. <https://leetcode.com/problems/string-to-integer-atoi/>

The algorithm consists of three parts.

In the first part we open the loop and move from left to right until we find a non empty symbol. Each time in the loop the variable startNumber is incremented by one.

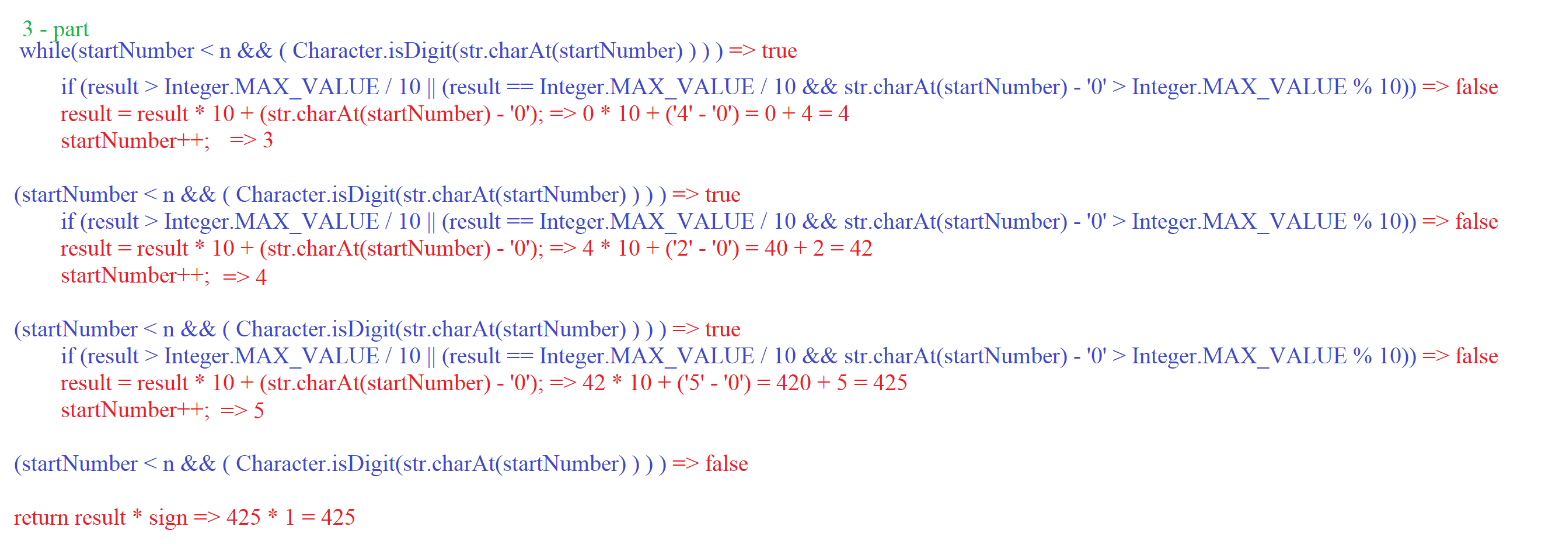


In the second case we define the number negative or positive. Taking the value with the startNumber index from str. If str.charAt(startNumber) is ‘-’ that sign be -1, else if str.charAt(startNumber) is “+” that sign be 1 and startNumber++.



In the third part, open the loop with the condition startNumber < n && ( Character.isDigit(str.charAt(startNumber) ) ). Each time we check for a condition (result > Integer.MAX\_VALUE / 10 || (result == Integer.MAX\_VALUE / 10 && str.charAt(startNumber) - '0' > Integer.MAX\_VALUE % 10)) and doing result = result \* 10 + (str.charAt(startNumber) - '0'), also startNumber++;

Return result \* sign;



**Java Code**

class Solution {

public int myAtoi(String str) {

int startNumber = 0;

int sign = 1;

int result = 0;

int n = str.length();

if(str.isEmpty())

return 0;

while(startNumber < n && str.charAt(startNumber) == ' ')

startNumber++;

if (startNumber < n && (str.charAt(startNumber) == '+' || str.charAt(startNumber) == '-'))

sign = (str.charAt(startNumber++) == '-') ? -1 : 1;

while(startNumber < n && ( Character.isDigit(str.charAt(startNumber) ) ) ) {

if (result > Integer.MAX\_VALUE / 10 ||

(result == Integer.MAX\_VALUE / 10 && str.charAt(startNumber) - '0' > Integer.MAX\_VALUE % 10)) {

return (sign == 1) ? Integer.MAX\_VALUE : Integer.MIN\_VALUE;

}

result = result \* 10 + (str.charAt(startNumber) - '0');

startNumber++;

}

return result \* sign;

}

}

1. <https://leetcode.com/problems/integer-to-english-words/>

**Description**

**273. Integer to English Words**

Convert a non-negative integer to its english words representation. Given input is guaranteed to be less than 2^31 - 1.

**Example 1:**

**Input:** 123

**Output:** "One Hundred Twenty Three"

**Example 2:**

**Input:** 12345

**Output:** "Twelve Thousand Three Hundred Forty Five"

**Example 3:**

**Input:** 1234567

**Output:** "One Million Two Hundred Thirty Four Thousand Five Hundred Sixty Seven"

**Example 4:**

**Input:** 1234567891

**Output:** "One Billion Two Hundred Thirty Four Million Five Hundred Sixty Seven Thousand Eight Hundred Ninety One

**Solution**

First we need to create three arrays.

* beforeTwenty = { "", "One", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Eleven", "Twelve", "Thirteen", "Fourteen", "Fifteen", "Sixteen", "Seventeen", "Eighteen", "Nineteen" };
* tens = {"", "Ten", "Twenty", "Thirty", "Forty", "Fifty", "Sixty","Seventy", "Eighty", "Ninety"};
* thousands = {"", "Thousand", "Million", "Billion"};

Opening the while loop with the condition num > 0. Each time we find a module from (nums % 1000) and check if (num % 1000 != 0) that call function findWord(nums % 1000). After do num /= 1000.

The findWord() function return word:

if(num < 20) {

return beforeTwenty[num];

} else if (num < 100) {

str = tens[num / 10];

str = (beforeTwenty[num % 10] != "") ? (str + " " + beforeTwenty[num % 10]) : str;

return str;

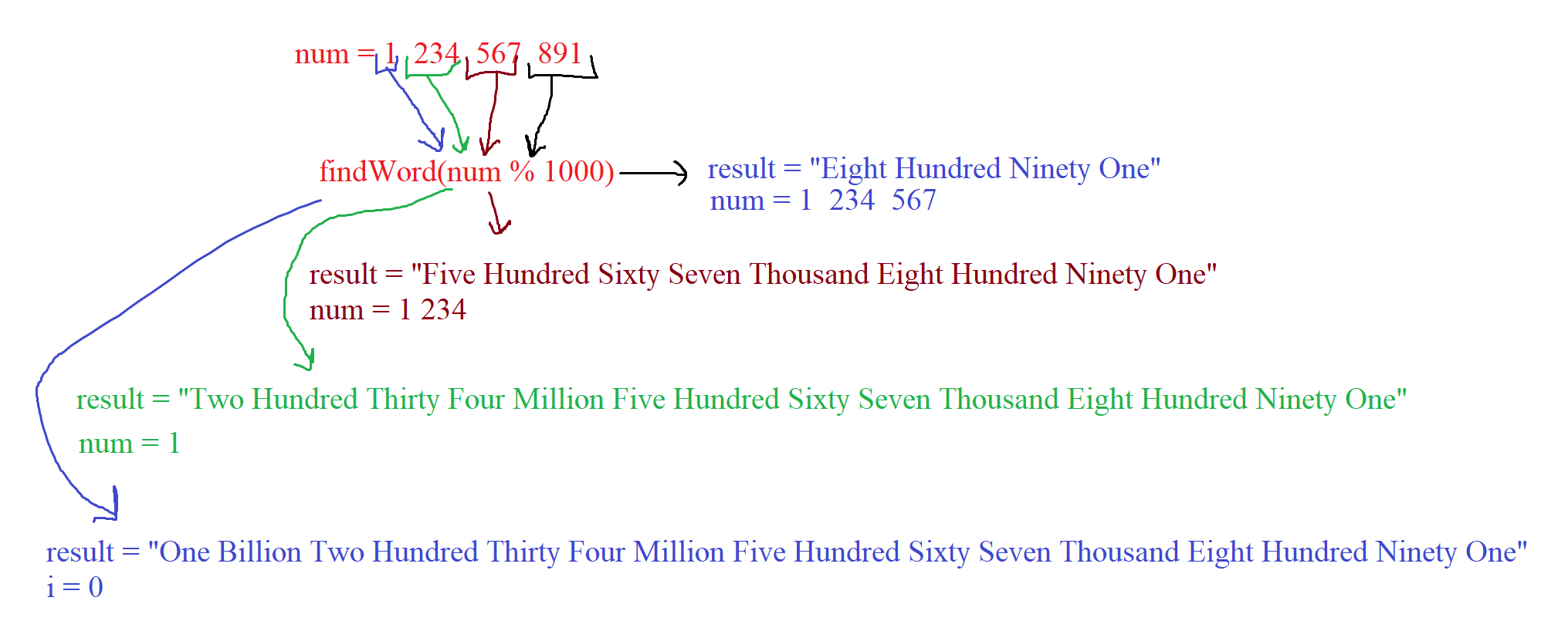
} else {

str = beforeTwenty[num / 100] + " Hundred";

str = (findWord(num % 100) != "") ? (str + " " + findWord(num % 100)) : str;

return str;

}

****

**Java Code**

class Solution {

private final static String[] beforeTwenty = { "", "One", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine",

"Ten", "Eleven", "Twelve", "Thirteen", "Fourteen", "Fifteen", "Sixteen", "Seventeen", "Eighteen", "Nineteen"

};

private final static String[] tens = {"", "Ten", "Twenty", "Thirty", "Forty", "Fifty", "Sixty","Seventy", "Eighty", "Ninety"};

private final String[] thousands = {"", "Thousand", "Million", "Billion"};

public String numberToWords(int num) {

String result = "";

String str = "";

int devideCurn = 0;

if(num == 0)

return "Zero";

while(num > 0) {

if(num % 1000 != 0){

str = findWord(num % 1000);

str = (thousands[devideCurn] != "") ? (str + " " + thousands[devideCurn]):str;

result = (result != "") ? (str + " " + result) : str;

str="";

}

num /= 1000;

devideCurn++;

}

return result;

}

public String findWord(int num) {

String str = "";

if(num < 20) {

return beforeTwenty[num];

} else if (num < 100) {

str = tens[num / 10];

str = (beforeTwenty[num % 10] != "") ? (str + " " + beforeTwenty[num % 10]) : str;

return str;

} else {

str = beforeTwenty[num / 100] + " Hundred";

str = (findWord(num % 100) != "") ? (str + " " + findWord(num % 100)) : str;

return str;

}

}

}

1. <https://leetcode.com/problems/reverse-words-in-a-string-iii/>

**Description**

**557. Reverse Words in a String III**

Given a string, you need to reverse the order of characters in each word within a sentence while still preserving whitespace and initial word order.

**Example 1:**

**Input:** "Let's take LeetCode contest"

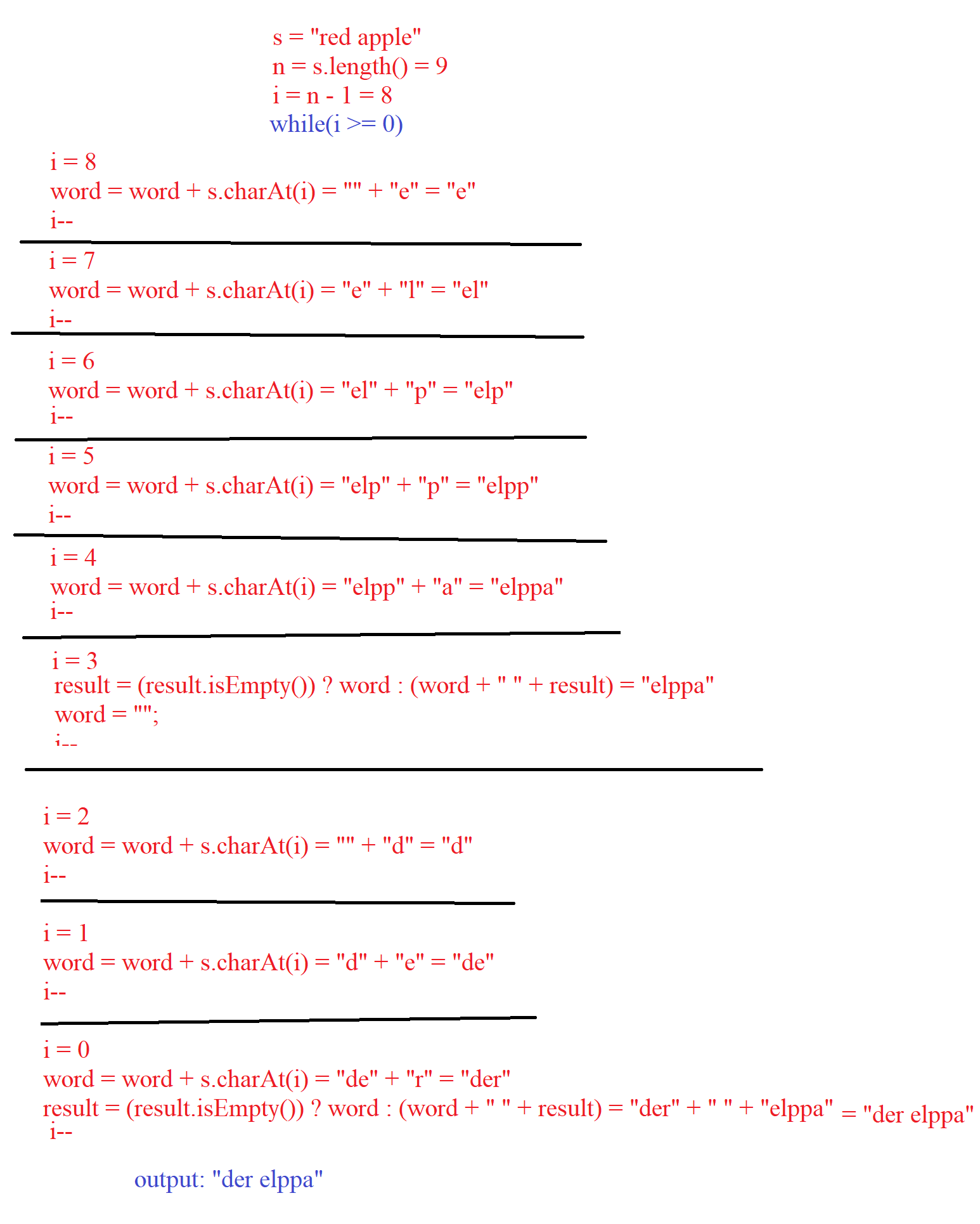
**Output:** "s'teL ekat edoCteeL tsetnoc"

**Solution**

Creating two string variables to store the result and a temporary reverse word. Assign the value (s.length() – 1) to the variable i. Open the loop and run until the variable i becomes (i >= 0). Each time in the loop, we check s.charAt(i):

* If s.charAt(i) != ‘ ’, doing word += s.charAt(i).
* If s.charAt(i) == ‘ ’ || i == 0, doing result = (result.isEmpty()) ? word : (word + " " + result) and word = "";

And each time after check s.charAt(i), doing i--.



**Java Code**

class Solution {

public String reverseWords(String s) {

int n = s.length();

String word = "", result = "";

int i = n - 1;

while(i >= 0) {

if(s.charAt(i) != ' ')

word += s.charAt(i);

if(s.charAt(i) == ' ' || i == 0) {

result = (result.isEmpty()) ? word : (word + " " + result);

word = "";

}

i--;

}

return result;

}

}