Write a function that adds two numbers. The catch, however, is that the numbers will be strings.

Examples

```
addStrNums("4", "5") ➞ "9"

addStrNums("abcdefg", "3") ➞ "-1"

addStrNums("1", "") ➞ "1"

addStrNums("1874682736267235927359283579235789257", "32652983572985729") ➞ "187
```

Notes

- If there are any non-numerical characters, return `"-1"`.
- An empty parameter should be treated as `"0"`.
- Your function should be able to add any size number.
- Your function doesn't have to add negative numbers.
- Zeros at the beginning of the string should be trimmed.
- **Bonus:** Don't use `BigInteger` or `BigDecimal` classes.

Create a function that takes a number as an argument and returns `true` if the number is a valid credit card number, `false` otherwise.

Credit card numbers must be between 14-19 digits in length, and pass the Luhn test, described below:

1. Remove the last digit (this is the "check digit").
2. Reverse the number.
3. Double the value of each digit in odd-numbered positions. If the doubled value has more than 1 digit, add the digits together (e.g. 8 x 2 = 16 ➞ 1 + 6 = 7).
4. Add all digits.
5. Subtract the last digit of the sum (from step 4) from 10. The result should be equal to the check digit from step 1.

Examples

```
validateCard(1234567890123456) ➞ false

// Step 1: check digit = 6, num = 123456789012345
// Step 2: num reversed = 543210987654321
// Step 3: digit array after selective doubling: [1, 4, 6, 2, 2, 0, 9, 8, 5, 6,
// Step 4: sum = 58
// Step 5: 10 - 8 = 2 (not equal to 6) ➞ false

validateCard(1234567890123452) ➞ true
```

You are given two strings `s` and `t`.

String `t` is generated by randomly shuffling string `s` and then adding one more letter at a random position.

Return the letter that was added to `t`.

Examples

```
findTheDifference("bcefg", "bcdefg") ➞ 'd'

findTheDifference("abcd", "abcde") ➞ 'e'

findTheDifference("aiou", "aeiou") ➞ 'e'

findTheDifference("mnoqrst", "mnopqrst") ➞ 'p'
```

Given two strings, `s1` and `s2`, select only the characters in each string where the character in the same position in the **other** string is in uppercase. Return these as a single string.

To illustrate, given the strings `s1 = "heLLo"` and `s2 = "GUlp"`, we select the letters "he" from `s1`, because "G" and "U" are uppercase. We then select "lp" from `s2`, because "LL" is in uppercase. Finally, we join these together and return `"help"`.

Examples

```
selectLetters("heLLO", "GUlp") ➞ "help"

selectLetters("1234567", "XxXxX")  ➞ "135"

selectLetters("EVERYTHING", "SomeThings") ➞  "EYSomeThings"
```

Notes

Create a function which validates whether a given array **alternates** between *positive* and *negative* numbers.

Examples

```
alternateSign([3, -2, 5, -5, 2, -8]) ➞ true

alternateSign([-6, 1, -1, 4, -3]) ➞ true

alternateSign([4, 4, -2, 3, -6, 10]) ➞ false
```

Notes

- Lists can be of any length.
- It doesn't matter if an array begins/ends with a positive or negative, as long as it alternates.
- If a list contains 0, return `false` (as it is neither positive nor negative).

Create a method that takes an array of integers and returns a new array, sorted in ascending order (smallest to biggest).

- Sort integer array in ascending order.
- If the function's argument is `null`, an empty array, or `undefined`; return an empty array.
- Return a new array of sorted numbers.

Examples

```
sortNumsAscending([1, 2, 10, 50, 5]) ➞ [1, 2, 5, 10, 50]

sortNumsAscending([80, 29, 4, -95, -24, 85]) ➞ [-95, -24, 4, 29, 80, 85]

sortNumsAscending(null) ➞ []

sortNumsAscending([]) ➞ []
```

Create a method that takes a string as its argument and returns the string in reversed order.

Examples

```
reverse("Hello World") ➞ "dlroW olleH"

reverse("The quick brown fox.") ➞ ".xof nworb kciuq ehT"

reverse("Edabit is really helpful!") ➞ "!lufpleh yllaer si tibadE"
```

Write a function that takes an array of elements and returns only the ints.

Examples

```
returnInts([9, 2, "space", "car", "lion", 16]) ➞ [9, 2, 16]

returnInts(["hello", 81, "basketball", 123, "fox"]) ➞ [81, 123]

returnInts([10, "121", 56, 20, "car", 3, "lion"]) ➞ [10, 56, 20, 3]

returnInts(["String",  true,  3.3,  1]) ➞ [1]
```

Create a function that takes a string and returns a string ordered by the length of the words. From shortest to longest word. If there are words with the same amount of letters, order them alphabetically.

Examples

```
sortByLength("Hello my friend") ➞ "my Hello friend"

sortByLength("Have a wonderful day") ➞ "a day Have wonderful"

sortByLenght("My son loves pineapples") ➞ "My son loves pineapples"
```

In this challenge, the goal is to calculate what time it is in two different cities. You're given a string `cityA` and the related string `timestamp` (time in `cityA`) with the date formatted in full **U.S. notation**, as in this example:

```
"July 21, 1983 23:01"
```

You have to return a new timestamp with date and corresponding time in `cityB`, formatted as in this example:

```
"1983-7-22 23:01"
```

See the table below for a list of given cities and their **GMT** (*Greenwich Mean Time*) hours offsets.

GMT	City
- 08:00	Los Angeles
- 05:00	New York
- 04:30	Caracas
- 03:00	Buenos Aires
00:00	London
+ 01:00	Rome
+ 03:00	Moscow
+ 03:30	Tehran
+ 05:30	New Delhi
+ 08:00	Beijing
+ 10:00	Canberra

Examples

```
timeDifference("Los Angeles", "April 1, 2011 23:23", "Canberra") ➞ "2011-4-2 17
// Can be a new day.

timeDifference("London", "July 31, 1983 23:01", "Rome") ➞ "1983-8-1 00:01"
// Can be a new month.

timeDifference("New York", "December 31, 1970 13:40", "Beijing") ➞ "1971-1-1 02
// Can be a new year.
```

Rearrange the Sentence

Published by Deep Xavier in Java

algorithms logic sorting strings

Given a sentence with numbers representing a word's location embedded within each word, return the sorted sentence.

Examples

```
rearrange("Tesh3 th5e 1I lov2e way6 she7 j4ust i8s.") ➞ "I love Tesh just the w

rearrange("the4 t3o man5 Happ1iest of6 no7 birt2hday steel8!") ➞ "Happiest birt

rearrange("is2 Thi1s T4est 3a") ➞ "This is a Test"

rearrange("4of Fo1r pe6ople g3ood th5e the2") ➞ "For the good of the people"

rearrange(" ") ➞ ""
```

Notes

Only the integers 1-9 will be used.

Add Two String Numbers

Published by ChirpyBoat in Java

language_fundamentals math numbers strings

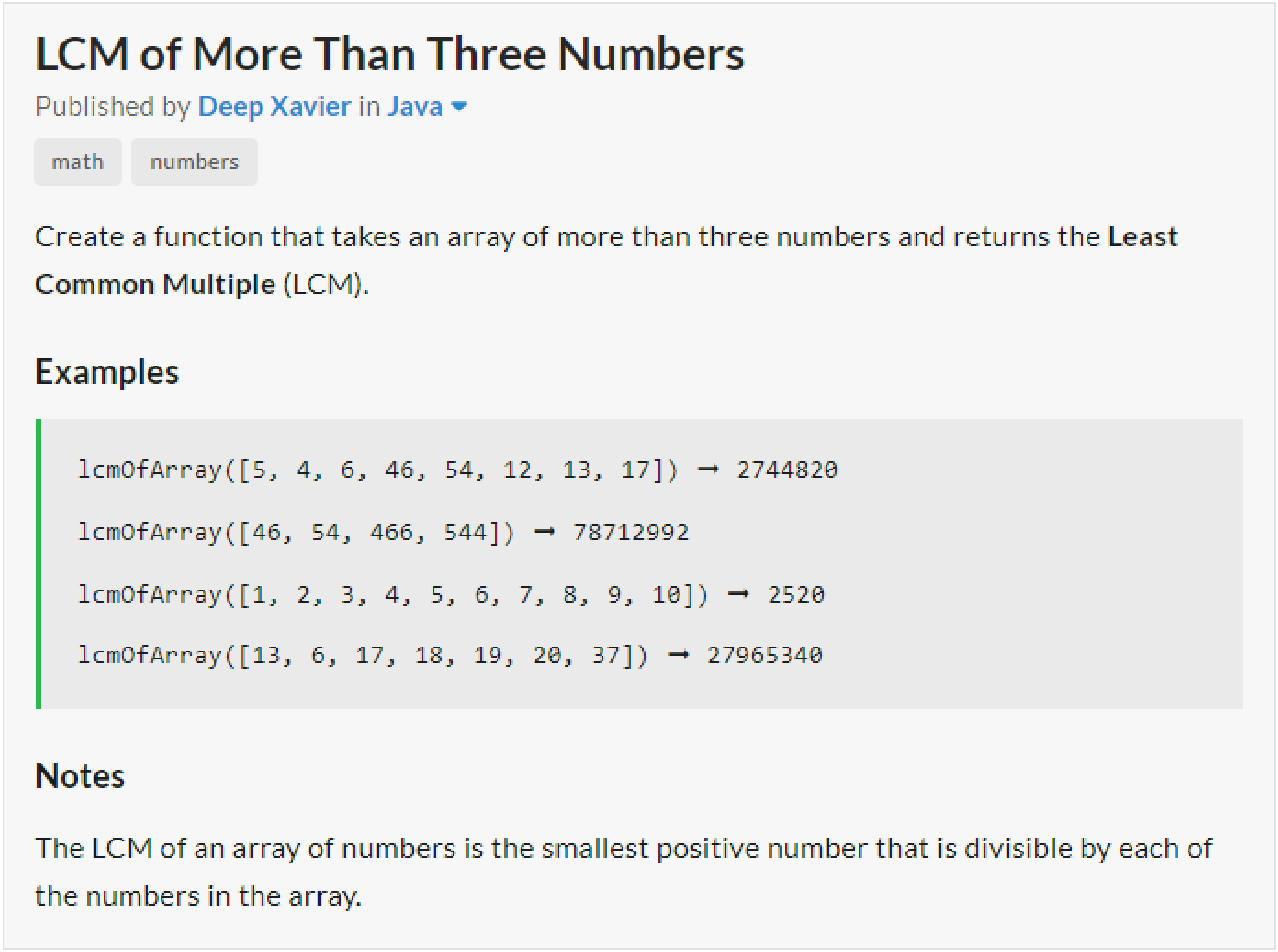
Write a function that adds two numbers. The catch, however, is that the numbers will be strings.

Examples

```
addStrNums("4", "5") ➞ "9"

addStrNums("abcdefg", "3") ➞ "-1"

addStrNums("1", "") ➞ "1"

addStrNums("1874682736267235927359283579235789257", "32652983572985729") ➞ "187
```

Notes

- If there are any non-numerical characters, return `"-1"`.
- An empty parameter should be treated as `"0"`.
- Your function should be able to add any size number.
- Your function doesn't have to add negative numbers.
- Zeros at the beginning of the string should be trimmed.
- **Bonus:** Don't use `BigInteger` or `BigDecimal` classes.



LCM of More Than Three Numbers

Published by Deep Xavier in Java

math numbers

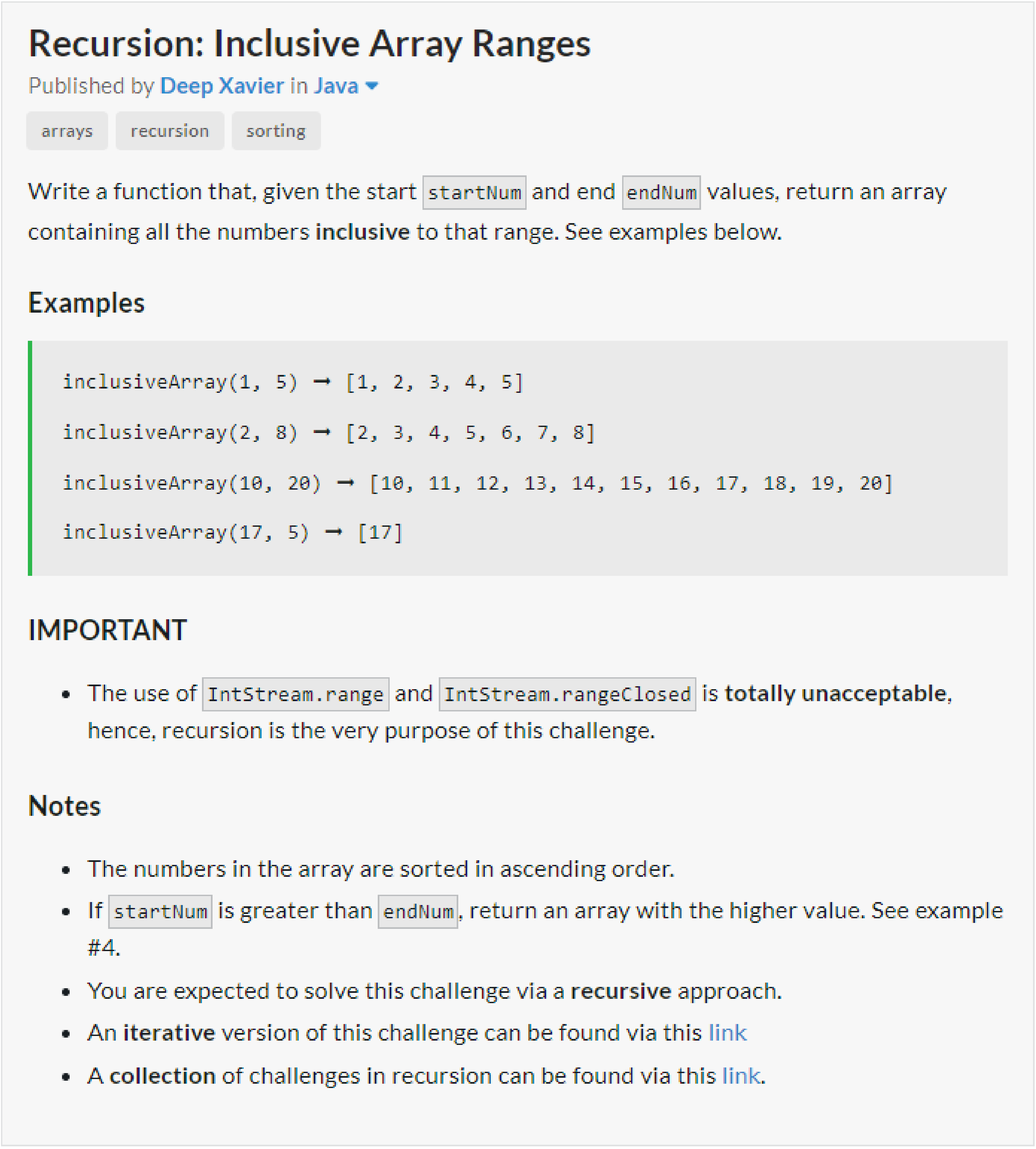
Create a function that takes an array of more than three numbers and returns the **Least Common Multiple** (LCM).

Examples

```
lcmOfArray([5, 4, 6, 46, 54, 12, 13, 17]) ➞ 2744820

lcmOfArray([46, 54, 466, 544]) ➞ 78712992

lcmOfArray([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]) ➞ 2520

lcmOfArray([13, 6, 17, 18, 19, 20, 37]) ➞ 27965340
```

Notes

The LCM of an array of numbers is the smallest positive number that is divisible by each of the numbers in the array.



Recursion: Inclusive Array Ranges

Published by Deep Xavier in Java

arrays recursion sorting

Write a function that, given the start `startNum` and end `endNum` values, return an array containing all the numbers **inclusive** to that range. See examples below.

Examples

```
inclusiveArray(1, 5) ➞ [1, 2, 3, 4, 5]

inclusiveArray(2, 8) ➞ [2, 3, 4, 5, 6, 7, 8]

inclusiveArray(10, 20) ➞ [10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]

inclusiveArray(17, 5) ➞ [17]
```

IMPORTANT

- The use of `IntStream.range` and `IntStream.rangeClosed` is **totally unacceptable**, hence, recursion is the very purpose of this challenge.

Notes

- The numbers in the array are sorted in ascending order.
- If `startNum` is greater than `endNum`, return an array with the higher value. See example #4.
- You are expected to solve this challenge via a **recursive** approach.
- An **iterative** version of this challenge can be found via this link
- A **collection** of challenges in recursion can be found via this link.

Strong Password

Published by Edward Clark in Java

strings validation

Create a function that determines the minimum number of characters needed to make a strong password.

A password is considered *strong* if it satisfies the following criteria:

- Its length is at least 6.
- It contains at least one digit.
- It contains at least one lowercase English character.
- It contains at least one uppercase English character.
- It contains at least one special character: `!@#$%^&*()-+`

Types of characters in a form you can paste into your solution:

```
static final String numbers = "0123456789";
static final String lower_case = "abcdefghijklmnopqrstuvwxyz";
static final String upper_case = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
static final String special_characters = "!@#$%^&*()-+";
```

Examples

```
strongPassword(“Ed1”) ➞ 3

strongPassword(“#Edabit”) ➞ 1

strongPassword("W1llth!spass?") ➞ 0
```

Notes

Try solving this without RegEx.

Disarium Number

Published by Deep Xavier in Java

loops math numbers strings validation +

A number is said to be Disarium if the **sum** of its *digits raised to their respective positions* is the number itself.

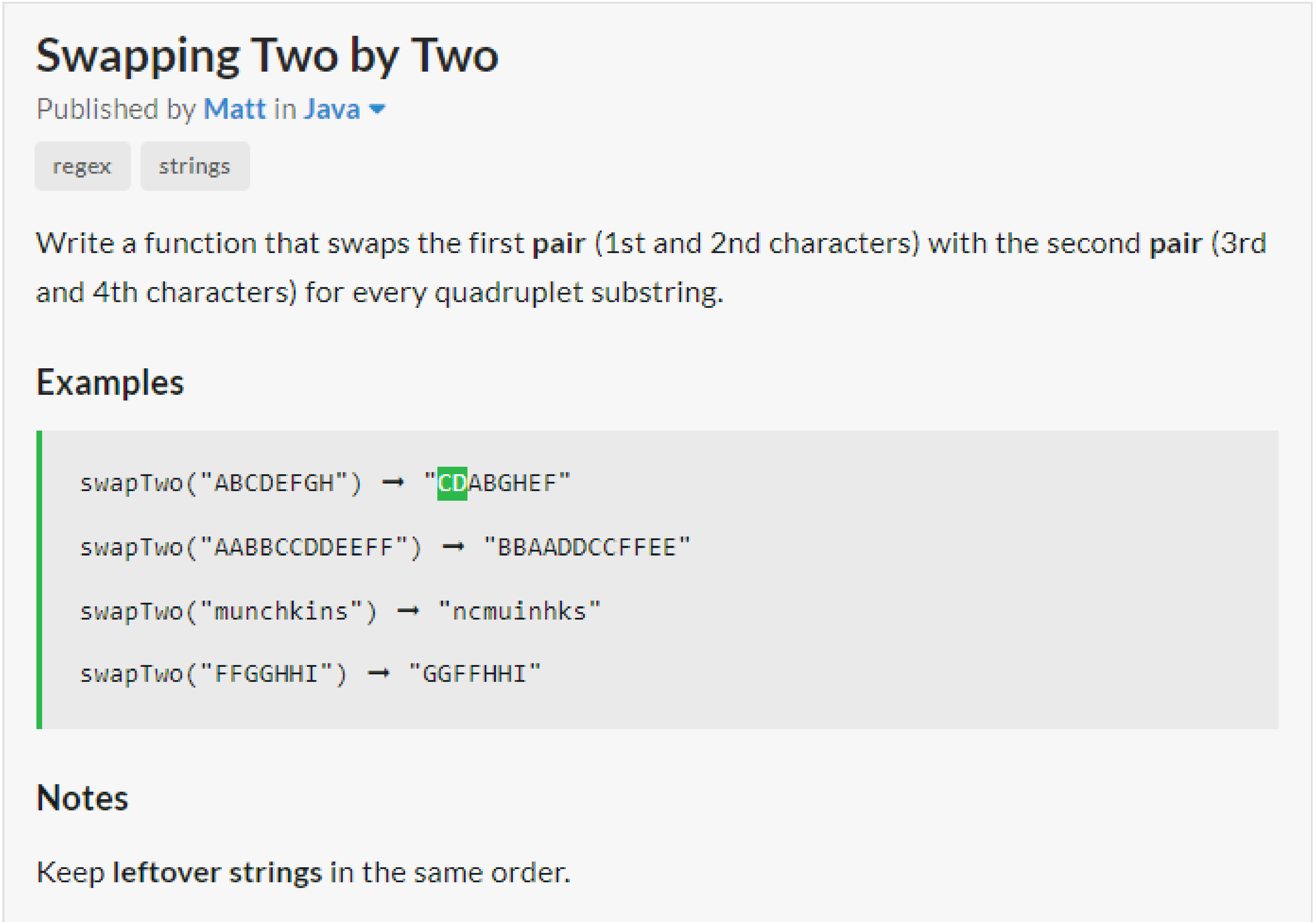
Create a function that determines whether a number is a Disarium or not.

Examples

```
isDisarium(75) ➞ false
// 7^1 + 5^2 = 7 + 25 = 32

isDisarium(135) ➞ true
// 1^1 + 3^2 + 5^3 = 1 + 9 + 125 = 135

isDisarium(544) ➞ false

isDisarium(518) ➞ true

isDisarium(8) ➞ true

isDisarium(466) ➞ false
```

Notes

- Position of the digit is 1-indexed.
- A recursive version of this challenge can be found via this link.



Swapping Two by Two

Published by Matt in Java

regex strings

Write a function that swaps the first **pair** (1st and 2nd characters) with the second **pair** (3rd and 4th characters) for every quadruplet substring.

Examples

```
swapTwo("ABCDEFGH") ➞ "CDABGHEF"

swapTwo("AABBCCDDEEFF") ➞ "BBAADDCCFFEE"

swapTwo("munchkins") ➞ "ncmuinhks"

swapTwo("FFGGHHI") ➞ "GGFFHHI"
```

Notes

Keep **leftover strings** in the same order.

Working 9 to 5

Published by Edward Clark in Java

algebra arrays math numbers

Write a function that calculates overtime and pay associated with overtime.

- Working 9 to 5: regular hours
- After 5pm is overtime

Your function gets an array with 4 values:

- Start of working day, in decimal format, (24-hour day notation)
- End of working day. (Same format)
- Hourly rate
- Overtime multiplier

Your function should spit out:

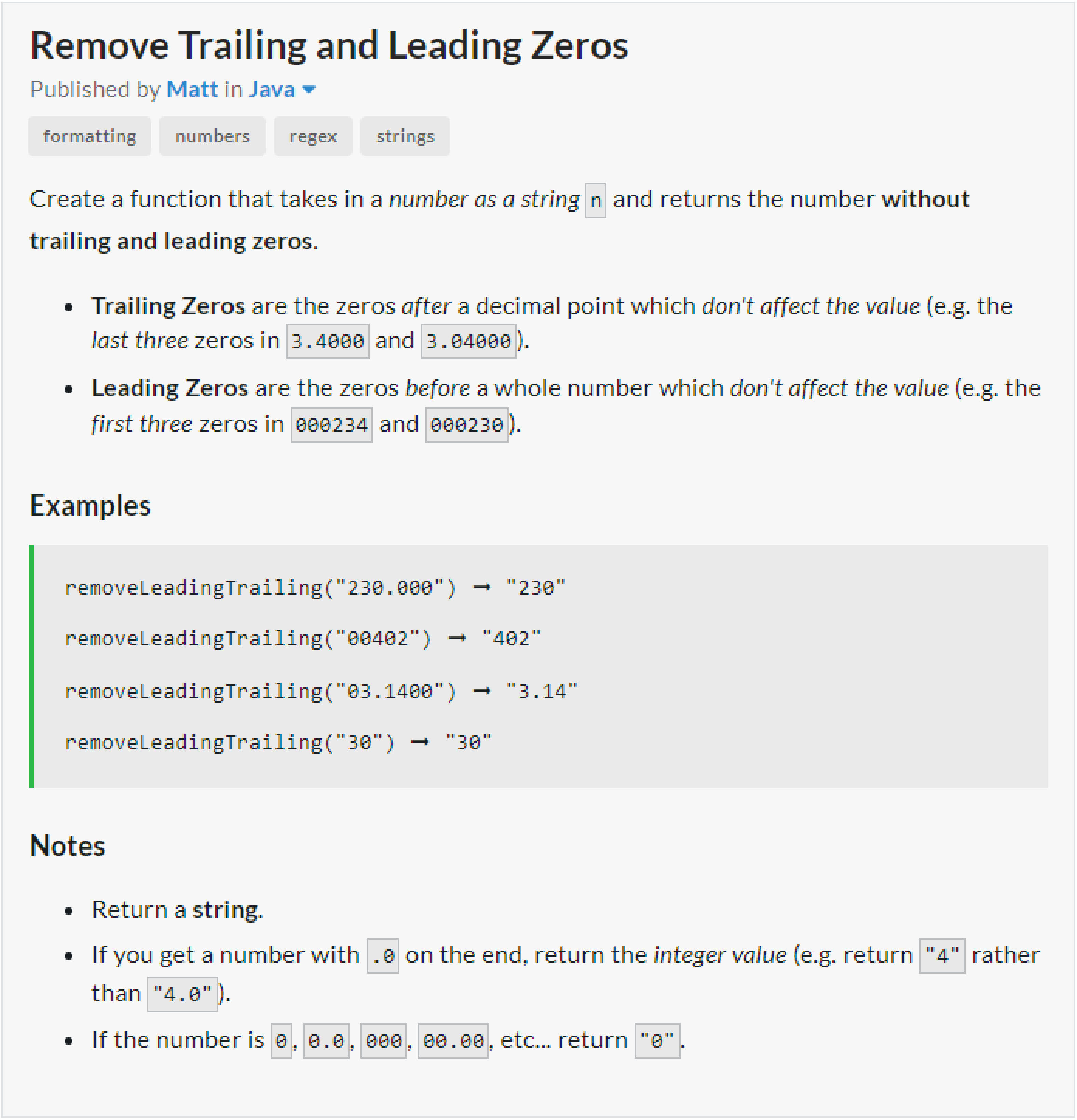
- `$` + earned that day (rounded to the nearest hundreth)

Examples

```
overTime([9, 17, 30, 1.5]) ➞ "$240.00"

overTime([16, 18, 30, 1.8]) ➞ "$84.00"

overTime([13.25, 15, 30, 1.5]) ➞ "$52.50"
```

2nd example explained:

- From 16 to 17 is regular, so `1 * 30` = 30
- From 17 to 18 is overtime, so `1 * 30 * 1.8` = 54
- `30 + 54` = $84.00



Remove Trailing and Leading Zeros

Published by Matt in Java

formatting numbers regex strings

Create a function that takes in a *number as a string* `n` and returns the number **without trailing and leading zeros.**

- **Trailing Zeros** are the zeros *after* a decimal point which *don't affect the value* (e.g. the *last three* zeros in `3.4000` and `3.04000`).
- **Leading Zeros** are the zeros *before* a whole number which *don't affect the value* (e.g. the *first three* zeros in `000234` and `000230`).

Examples

```
removeLeadingTrailing("230.000") ➞ "230"

removeLeadingTrailing("00402") ➞ "402"

removeLeadingTrailing("03.1400") ➞ "3.14"

removeLeadingTrailing("30") ➞ "30"
```

Notes

- Return a **string**.
- If you get a number with `.0` on the end, return the *integer value* (e.g. return `"4"` rather than `"4.0"`).
- If the number is `0`, `0.0`, `000`, `00.00`, etc... return `"0"`.

Print Grid

Published by MrDrProfessor in Java

language_fundamentals loops +

Write a method that accepts two integer parameters rows and cols. The output is a comma-separated grid of numbers displayed in column-major order, meaning the numbers shown increase sequentially down each column and wrap to the top of the next column to the right once the bottom of the current column is reached.

Examples

```
printGrid(3, 6) ➞ "
  1, 4, 7, 10, 13, 16
  2, 5, 8, 11, 14, 17
  3, 6, 9, 12, 15, 18
"

printGrid(5, 3) ➞ "
  1, 6, 11
  2, 7, 12
  3, 8, 13
  4, 9, 14
  5, 10, 15
"

printGrid(4, 1) ➞ "
  1
  2
  3
  4
"
```

Notes

- Each row ends with "\n" e.g. "1, 6, 11\n" rather then a space "1, 6, 11 " (=wrong)
- Last line "\n" should be trimed.

12 vs 24 Hours

Published by Matt in Java

dates formatting strings

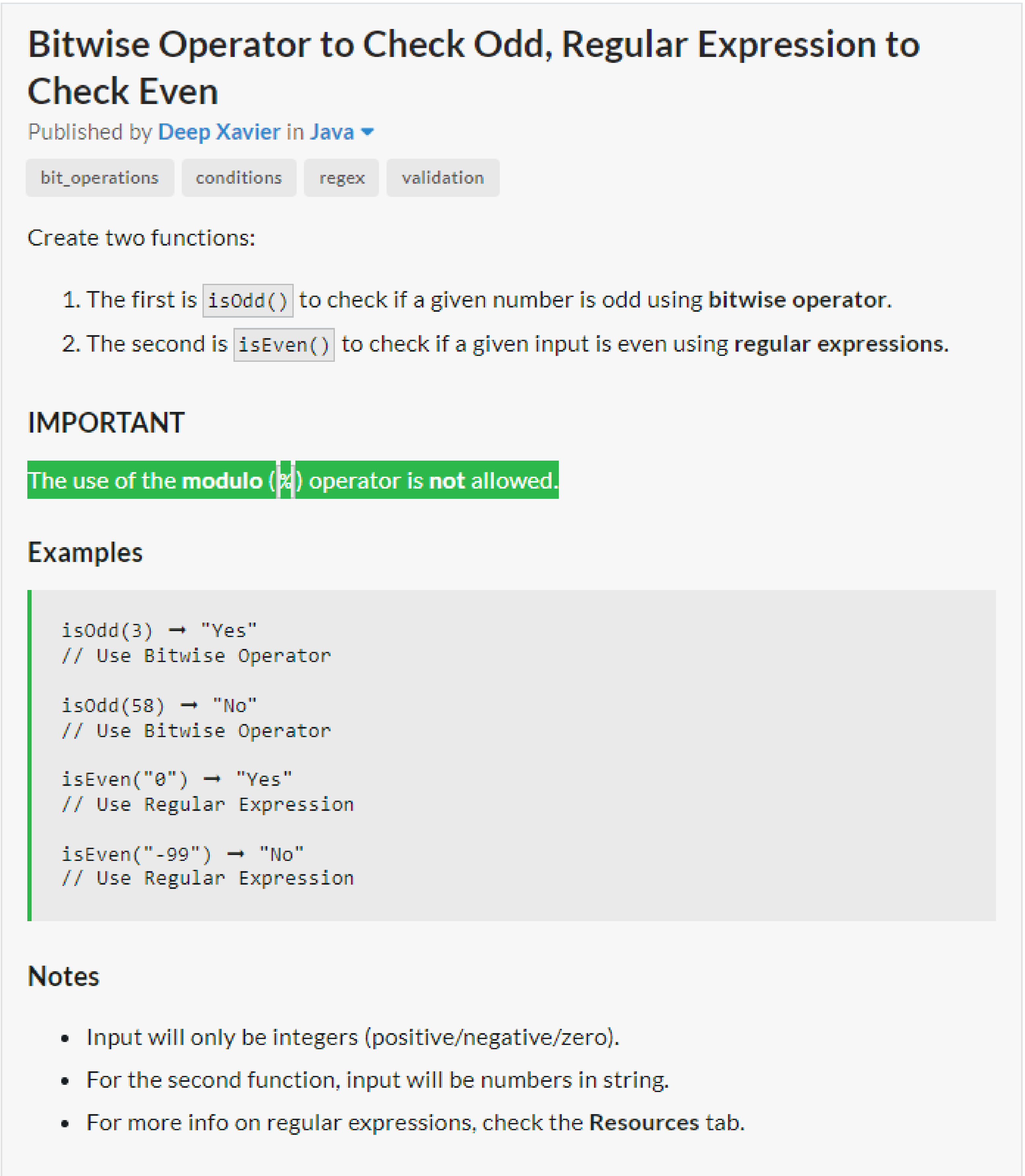
Create a function that converts 12-hour time to 24-hour time or vice versa. Return the output as a string.

Examples

```
convertTime("12:00 am") ➞ "0:00"

convertTime("6:20 pm") ➞ "18:20"

convertTime("21:00") ➞ "9:00 pm"

convertTime("5:05") ➞ "5:05 am"
```

Notes

- A 12-hour time input will be denoted with an **am** or **pm** suffix.
- A 24-hour input time contains no suffix.



Bitwise Operator to Check Odd, Regular Expression to Check Even

Published by Deep Xavier in Java

bit_operations conditions regex validation

Create two functions:

1. The first is `isOdd()` to check if a given number is odd using **bitwise operator**.
2. The second is `isEven()` to check if a given input is even using **regular expressions**.

IMPORTANT

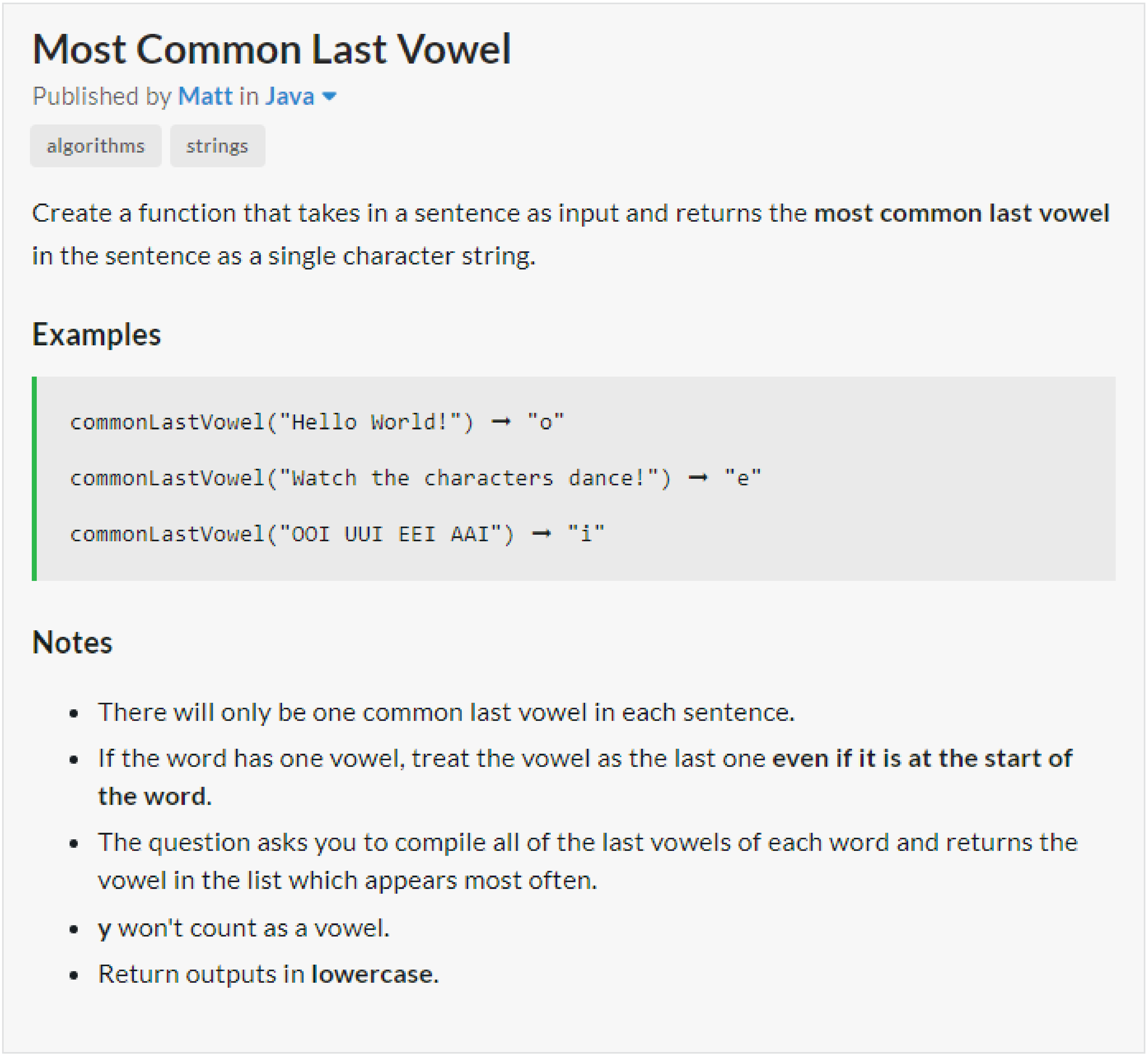
The use of the **modulo** (`%`) operator **is not** allowed.

Examples

```
isOdd(3) ➞ "Yes"
// Use Bitwise Operator

isOdd(58) ➞ "No"
// Use Bitwise Operator

isEven("0") ➞ "Yes"
// Use Regular Expression

isEven("-99") ➞ "No"
// Use Regular Expression
```

Notes

- Input will only be integers (positive/negative/zero).
- For the second function, input will be numbers in string.
- For more info on regular expressions, check the **Resources** tab.



Most Common Last Vowel

Published by Matt in Java

algorithms strings

Create a function that takes in a sentence as input and returns the **most common last vowel** in the sentence as a single character string.

Examples

```
commonLastVowel("Hello World!") ➞ "o"

commonLastVowel("Watch the characters dance!") ➞ "e"

commonLastVowel("OOI UUI EEI AAI") ➞ "i"
```

Notes

- There will only be one common last vowel in each sentence.
- If the word has one vowel, treat the vowel as the last one **even if it is at the start of the word.**
- The question asks you to compile all of the last vowels of each word and returns the vowel in the list which appears most often.
- **y** won't count as a vowel.
- Return outputs in **lowercase.**

Create a function which takes in an encoded string and returns an object according to the following example:

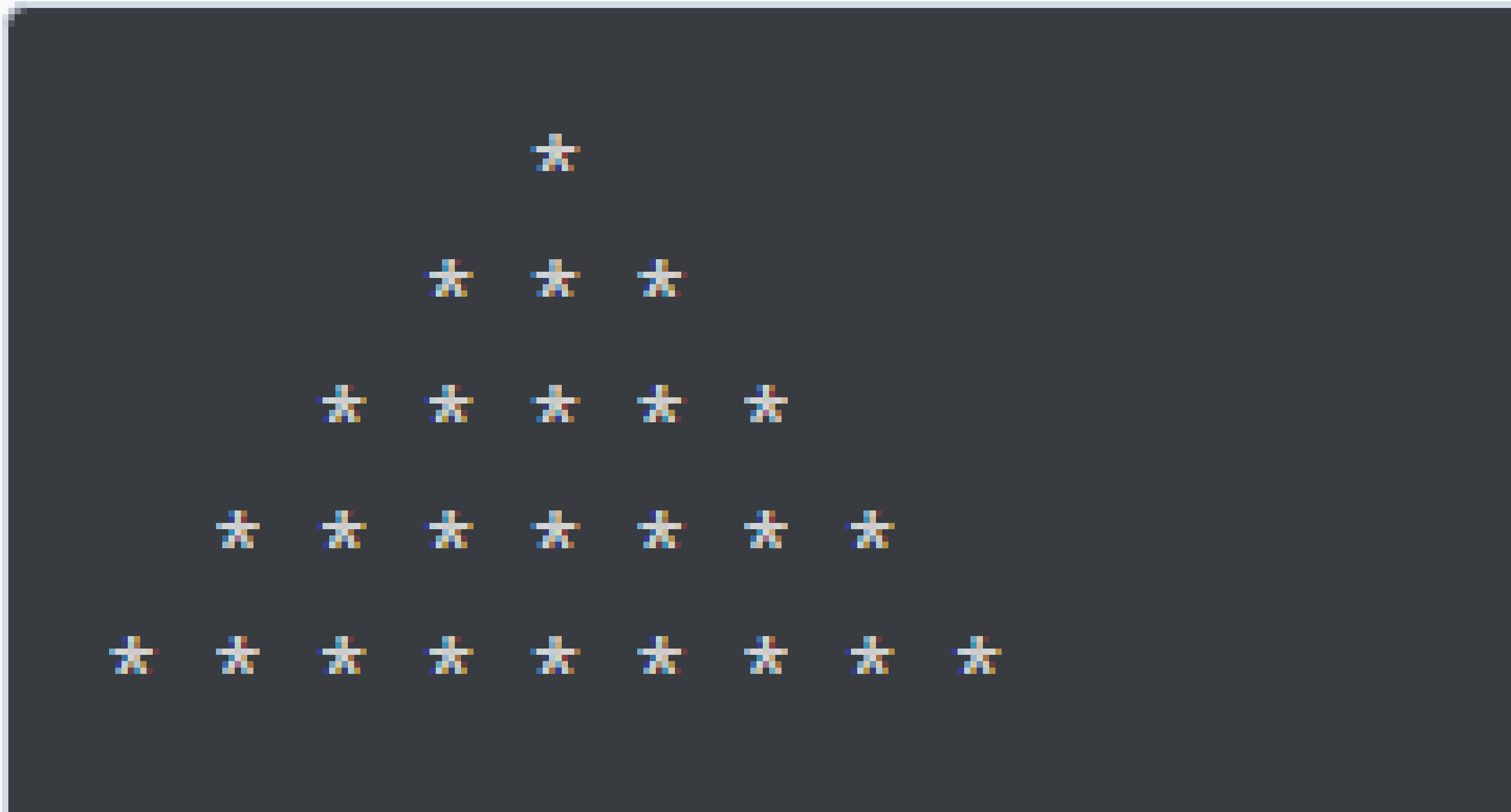
Examples

```
parseCode("Tesha000Deep00014344") ➞ {
  "firstName"="Tesha",
  "lastName"="Deep",
  "id"="14344"
}

parseCode("John000Doe000123") ➞ {
  "firstName"="John",
  "lastName"="Doe",
  "id"="123"
}

parseCode("kevin0smith004331") ➞ {
  "firstName"="kevin",
  "lastName"="smith",
  "id"="4331"
}

parseCode("Thomas00LEE0000043") ➞ {
  "firstName"="Thomas",
  "lastName"="LEE",
  "id"="43"
}

parseCode("Madsy0Jupiter0321") ➞ {
  "firstName"="Madsy",
  "lastName"="Jupiter",
  "id"="321"
}
```

Notes

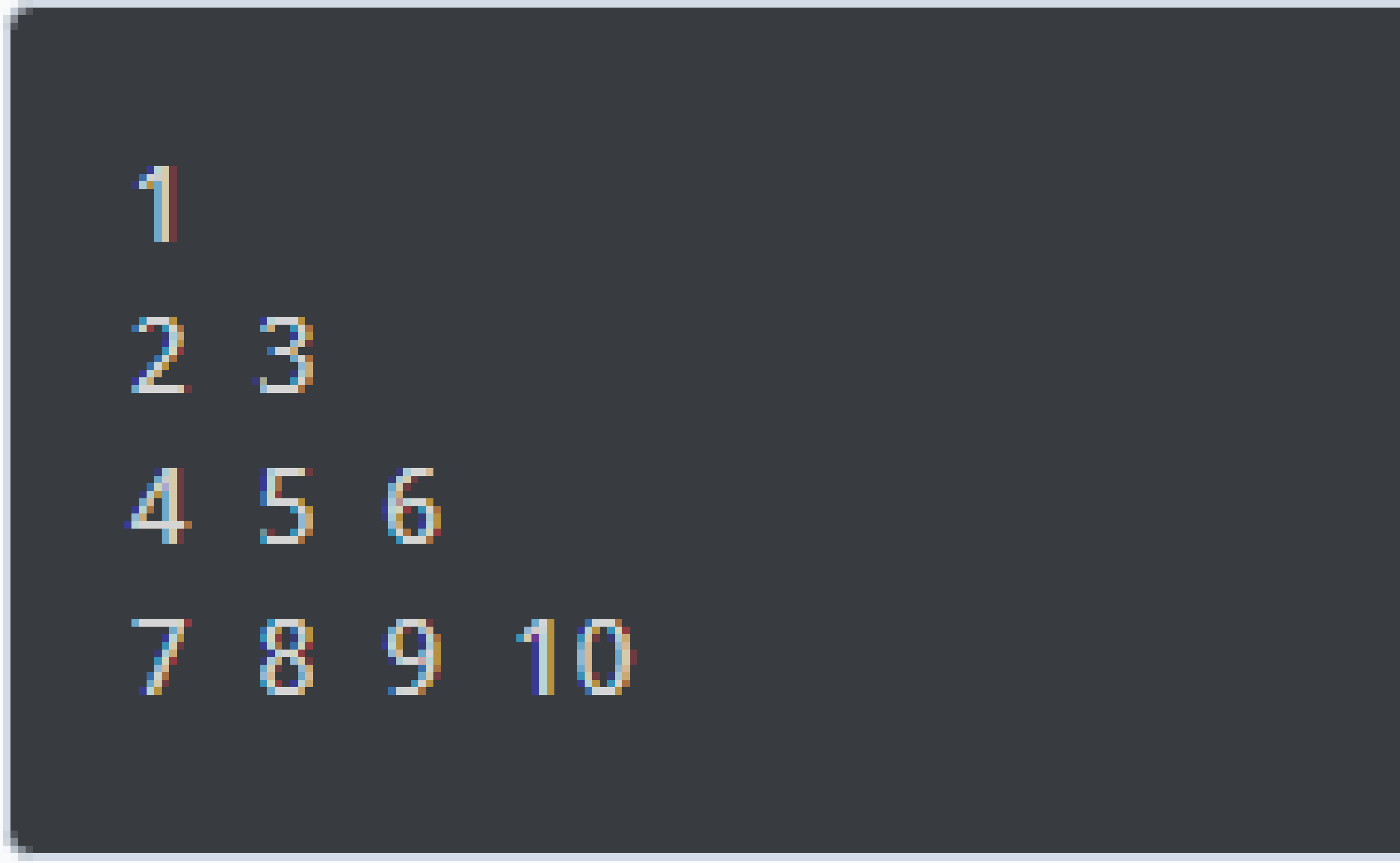
- The string will always be in the same format, first name, last name and id with zeros between them.
- `id` numbers will not contain any zeros.

```
*
* *
* * *
* * * *
* * * * *
```



```
        *

      * * *

    * * * * *

  * * * * * * *

* * * * * * * * *
```

```
          1
        1   1
      1   2   1
    1   3   3    1
  1   4   6   4   1
1   5   10   10  5   1
```



```
1
2 3
4 5 6
7 8 9 10
```