**1/6**

1. В Java есть единственный оператор, способный обеспечить остаток от операции деления. Два числа передаются в качестве параметров. Первый параметр, разделенный на второй параметр, будет иметь остаток, возможно, ноль. Верните это значение.

Пример:

remainder(1, 3) ➞ 1

remainder(3, 4) ➞ 3

remainder(-9, 45) ➞ -9

remainder(5, 5) ➞ 0

1. Напишите функцию, которая принимает основание и высоту треугольника и возвращает его площадь.

Пример:

triArea(3, 2) ➞ 3

triArea(7, 4) ➞ 14

triArea(10, 10) ➞ 50

1. В этой задаче фермер просит вас сказать ему, сколько ног можно сосчитать среди всех его животных. Фермер разводит три вида:

chickens = 2 legs

cows = 4 legs

pigs = 4 legs

Фермер подсчитал своих животных, и он дает вам промежуточный итог для каждого вида. Вы должны реализовать функцию, которая возвращает общее количество ног всех животных.

Пример:

animals(2, 3, 5) ➞ 36

animals(1, 2, 3) ➞ 22

animals(5, 2, 8) ➞ 50

1. Создайте функцию, которая принимает три аргумента (prob, prize, pay) и возвращает true, если prob \* prize > pay; в противном случае возвращает false.

Чтобы проиллюстрировать это: profitableGamble (0,2, 50, 9) должен выдать значение true, поскольку 1 (0,2 \* 50 - 9), а 1> 0.

Пример:

profitableGamble(0.2, 50, 9) ➞ true

profitableGamble(0.9, 1, 2) ➞ false

profitableGamble(0.9, 3, 2) ➞ true

1. Напишите функцию, которая принимает 3 числа и возвращает, что нужно сделать с a и b: они должны быть сложены, вычитаны, умножены или разделены, чтобы получить N. Если ни одна из операций не может дать N, верните "none".

3 числа – это N, a и b.

Пример:

operation(24, 15, 9) ➞ "added"

operation(24, 26, 2) ➞ "subtracted"

operation(15, 11, 11) ➞ "none"

Примечания:

– В качестве тестового ввода используются только целые числа.

– Числа должны быть добавлены, вычтены, разделены или умножены в порядке их появления в параметрах.

1. Создайте функцию, которая возвращает значение ASCII переданного символа.

Пример:

ctoa('A') ➞ 65

ctoa('m') ➞ 109

ctoa('[') ➞ 91

ctoa('\') ➞ 92

1. Напишите функцию, которая берет последнее число из последовательного списка чисел и возвращает сумму всех чисел до него и включая его.

Пример:

addUpTo(3) ➞ 6

// 1 + 2 + 3 = 6

addUpTo(10) ➞ 55

// 1 + 2 + 3 + ... + 10 = 55

addUpTo(7) ➞ 28

// 1 + 2 + 3 + ... + 7 = 28

1. Создайте функцию, которая находит максимальное значение третьего ребра треугольника, где длины сторон являются целыми числами.

Пример:

nextEdge(8, 10) ➞ 17

nextEdge(5, 7) ➞ 11

nextEdge(9, 2) ➞ 10

1. Создайте функцию, которая принимает массив чисел и возвращает сумму его кубов.

Пример:

sumOfCubes([1, 5, 9]) ➞ 855

// Since 1^3 + 5^3 + 9^3 = 1 + 125 + 729 = 855

sumOfCubes([3, 4, 5]) ➞ 216

sumOfCubes([2]) ➞ 8

sumOfCubes([]) ➞ 0

1. Создайте функцию, которая будет для данного a, b, c выполнять следующие действия:

– Добавьте A к себе B раз.

* Проверьте, делится ли результат на C.

Пример:

abcmath(42, 5, 10) ➞ false

// 42+42 = 84,84+84 = 168,168+168 = 336,336+336 = 672, 672+672 = 1344

// 1344 is not divisible by 10

abcmath(5, 2, 1) ➞ true

abcmath(1, 2, 3) ➞ false

**2/6**

1. Создайте функцию, которая повторяет каждый символ в строке n раз.

Пример:

repeat("mice", 5) ➞ "mmmmmiiiiiccccceeeee"

repeat("hello", 3) ➞ "hhheeellllllooo"

repeat("stop", 1) ➞ "stop"

1. Создайте функцию, которая принимает массив и возвращает разницу между самыми большими и самыми маленькими числами.

Пример:

differenceMaxMin([10, 4, 1, 4, -10, -50, 32, 21]) ➞ 82

// Smallest number is -50, biggest is 32.

differenceMaxMin([44, 32, 86, 19]) ➞ 67

// Smallest number is 19, biggest is 86.

1. Создайте функцию, которая принимает массив в качестве аргумента и возвращает true или false в зависимости от того, является ли среднее значение всех элементов массива целым числом или нет.

Пример:

isAvgWhole([1, 3]) ➞ true

isAvgWhole([1, 2, 3, 4]) ➞ false

isAvgWhole([1, 5, 6]) ➞ true

isAvgWhole([1, 1, 1]) ➞ true

isAvgWhole([9, 2, 2, 5]) ➞ false

1. Создайте метод, который берет массив целых чисел и возвращает массив, в котором каждое целое число является суммой самого себя + всех предыдущих чисел в массиве.

Пример:

cumulativeSum([1, 2, 3]) ➞ [1, 3, 6]

cumulativeSum([1, -2, 3]) ➞ [1, -1, 2]

cumulativeSum([3, 3, -2, 408, 3, 3]) ➞ [3, 6, 4, 412, 415, 418]

1. Создайте функцию, которая возвращает число десятичных знаков, которое имеет число (заданное в виде строки). Любые нули после десятичной точки отсчитываются в сторону количества десятичных знаков.

Пример:

getDecimalPlaces("43.20") ➞ 2

getDecimalPlaces("400") ➞ 0

getDecimalPlaces("3.1") ➞ 1

1. Создайте функцию, которая при заданном числе возвращает соответствующее число Фибоначчи.

Пример:

Fibonacci(3) ➞ 3

Fibonacci(7) ➞ 21

Fibonacci(12) ➞ 233

1. Почтовые индексы состоят из 5 последовательных цифр. Учитывая строку, напишите функцию, чтобы определить, является ли вход действительным почтовым индексом. Действительный почтовый индекс выглядит следующим образом:

– Должно содержать только цифры (не допускается использование нецифровых цифр).

– Не должно содержать никаких пробелов.

– Длина не должна превышать 5 цифр.

Пример:

isValid("59001") ➞ true

isValid("853a7") ➞ false

isValid("732 32") ➞ false

isValid("393939") ➞ false

1. Пара строк образует странную пару, если оба из следующих условий истинны:

– Первая буква 1-й строки = последняя буква 2-й строки.

– Последняя буква 1-й строки = первая буква 2-й строки.

– Создайте функцию, которая возвращает true, если пара строк представляет собой странную пару, и false в противном случае.

Пример:

isStrangePair("ratio", "orator") ➞ true

// "ratio" ends with "o" and "orator" starts with "o".

// "ratio" starts with "r" and "orator" ends with "r".

isStrangePair("sparkling", "groups") ➞ true

isStrangePair("bush", "hubris") ➞ false

isStrangePair("", "") ➞ true

1. Создайте две функции: isPrefix(word, prefix-) и isSuffix (word, -suffix).

– isPrefix должен возвращать true, если он начинается с префиксного аргумента.

– isSuffix должен возвращать true, если он заканчивается аргументом суффикса.

– В противном случае верните false.

Пример:

isPrefix("automation", "auto-") ➞ true

isSuffix("arachnophobia", "-phobia") ➞ true

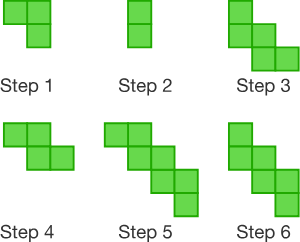
isPrefix("retrospect", "sub-") ➞ false

isSuffix("vocation", "-logy") ➞ false

Примечание:

Аргументы префикса и суффикса имеют тире - в них.

1. Создайте функцию, которая принимает число (шаг) в качестве аргумента и возвращает количество полей на этом шаге последовательности.



Шаг 0: начните с 0

Шаг 1: Добавьте 3

Шаг 2: Вычтите 1

Повторите Шаги 1 И 2 ...

Пример:

boxSeq(0) ➞ 0

boxSeq(1) ➞ 3

boxSeq(2) ➞ 2

**3/6**

1. Квадратное уравнение ax2 + bx + c = 0 имеет либо 0, либо 1, либо 2 различных решения для действительных значений x. учитывая a, b и c, вы должны вернуть число решений в уравнение.

Пример:

solutions(1, 0, -1) ➞ 2

// x² - 1 = 0 has two solutions (x = 1 and x = -1).

solutions(1, 0, 0) ➞ 1

// x² = 0 has one solution (x = 0).

solutions(1, 0, 1) ➞ 0

// x² + 1 = 0 has no solutions.

1. Напишите функцию, которая возвращает позицию второго вхождения " zip " в строку, или -1, если оно не происходит по крайней мере дважды. Ваш код должен быть достаточно общим, чтобы передать все возможные случаи, когда "zip" может произойти в строке.

Пример:

findZip("all zip files are zipped") ➞ 18

findZip("all zip files are compressed") ➞ -1

Примечание:

Верхний регистр " Zip "- это не то же самое, что нижний регистр "zip".

1. Создайте функцию, которая проверяет, является ли целое число совершенным числом или нет. Совершенное число - это число, которое можно записать как сумму его множителей, исключая само число.

Например, 6-это идеальное число, так как 1 + 2 + 3 = 6, где 1, 2 и 3-Все коэффициенты 6. Точно так же 28-это совершенное число, так как 1 + 2 + 4 + 7 + 14 = 28.

Пример:

checkPerfect(6) ➞ true

checkPerfect(28) ➞ true

checkPerfect(496) ➞ true

checkPerfect(12) ➞ false

checkPerfect(97) ➞ false

1. Создайте функцию, которая принимает строку и возвращает новую строку с заменой ее первого и последнего символов, за исключением трех условий:

– Если длина строки меньше двух, верните "несовместимо".".

– Если первый и последний символы совпадают, верните "два-это пара.".

Пример:

flipEndChars("Cat, dog, and mouse.") ➞ ".at, dog, and mouseC"

flipEndChars("ada") ➞ "Two's a pair."

flipEndChars("Ada") ➞ "adA"

flipEndChars("z") ➞ "Incompatible."

1. Создайте функцию, которая определяет, является ли строка допустимым шестнадцатеричным кодом.

Шестнадцатеричный код должен начинаться с фунтового ключа # и иметь длину ровно 6 символов. Каждый символ должен быть цифрой от 0-9 или буквенным символом от A-F. все буквенные символы могут быть прописными или строчными.

Пример:

isValidHexCode("#CD5C5C") ➞ true

isValidHexCode("#EAECEE") ➞ true

isValidHexCode("#eaecee") ➞ true

isValidHexCode("#CD5C58C") ➞ false

// Length exceeds 6

isValidHexCode("#CD5C5Z") ➞ false

// Not all alphabetic characters in A-F

isValidHexCode("#CD5C&C") ➞ false

// Contains unacceptable character

isValidHexCode("CD5C5C") ➞ false

// Missing #

1. Напишите функцию, которая возвращает true, если два массива имеют одинаковое количество уникальных элементов, и false в противном случае.

Для примера:

arr1 = [1, 3, 4, 4, 4]

arr2 = [2, 5, 7]

В arr1 число 4 появляется трижды, что означает, что оно содержит три уникальных элемента: [1, 3, 4]. Поскольку arr1 и arr2 содержат одинаковое количество уникальных элементов, этот пример вернет значение true.

Пример:

same([1, 3, 4, 4, 4], [2, 5, 7]) ➞ true

same([9, 8, 7, 6], [4, 4, 3, 1]) ➞ false

same([2], [3, 3, 3, 3, 3]) ➞ true

1. Число Капрекара-это положительное целое число, которое после возведения в квадрат и разбиения на две лексикографические части равно сумме двух полученных новых чисел:

– Если количество цифр квадратного числа четное, то левая и правая части будут иметь одинаковую длину.

– Если количество цифр квадратного числа нечетно, то правая часть будет самой длинной половиной, а левая-самой маленькой или равной нулю, если количество цифр равно 1.

– Учитывая положительное целое число n, реализуйте функцию, которая возвращает true, если это число Капрекара, и false, если это не так.

Пример:

isKaprekar(3) ➞ false

// n² = "9"

// Left + Right = 0 + 9 = 9 ➞ 9 !== 3

isKaprekar(5) ➞ false

// n² = "25"

// Left + Right = 2 + 5 = 7 ➞ 7 !== 5

isKaprekar(297) ➞ true

// n² = "88209"

// Left + Right = 88 + 209 = 297 ➞ 297 === 297

Примечание:

Тривиально, 0 и 1-Это числа Капрекара, являющиеся единственными двумя числами, равными их квадрату.

1. Напишите функцию, которая возвращает самую длинную последовательность последовательных нулей в двоичной строке.

Пример:

longestZero("01100001011000") ➞ "0000"

longestZero("100100100") ➞ "00"

longestZero("11111") ➞ ""

1. Если задано целое число, создайте функцию, которая возвращает следующее простое число. Если число простое, верните само число.

Пример:

nextPrime(12) ➞ 13

nextPrime(24) ➞ 29

nextPrime(11) ➞ 11

// 11 is a prime, so we return the number itself.

1. Учитывая три числа, x, y и z, определите, являются ли они ребрами прямоугольного треугольника.

Пример:

rightTriangle(3, 4, 5) ➞ true

// This is the classic example of a "nice" right angled triangle.

rightTriangle(145, 105, 100) ➞ true

// This is a less famous example.

rightTriangle(70, 130, 110) ➞ false

// This isn't a right angled triangle.

**4/6**

1. Бесси работает над сочинением для своего класса писателей. Поскольку ее почерк довольно плох, она решает напечатать эссе с помощью текстового процессора. Эссе содержит N слов (1≤N≤100), разделенных пробелами. Каждое слово имеет длину от 1 до 15 символов включительно и состоит только из прописных или строчных букв. Согласно инструкции к заданию, эссе должно быть отформатировано очень специфическим образом: каждая строка должна содержать не более K (1≤K≤80) символов, не считая пробелов. К счастью, текстовый процессор Бесси может справиться с этим требованием, используя следующую стратегию:

– Если Бесси набирает Слово, и это слово может поместиться в текущей строке, поместите его в эту строку. В противном случае поместите слово на следующую строку и продолжайте добавлять к этой строке. Конечно, последовательные слова в одной строке все равно должны быть разделены одним пробелом. В конце любой строки не должно быть места.

* К сожалению, текстовый процессор Бесси только что сломался. Пожалуйста, помогите ей правильно оформить свое эссе!

Вам будут даны n, k и строка

Пример:

10,7 hello my name is Bessie and this is my essay➞

hello my

name is

Bessie

and this

is my

essay

1. Напишите функцию, которая группирует строку в кластер скобок. Каждый кластер должен быть сбалансирован.

Пример:

split("()()()") ➞ ["()", "()", "()"]

split("((()))") ➞ ["((()))"]

split("((()))(())()()(()())") ➞ ["((()))", "(())", "()", "()", "(()())"]

split("((())())(()(()()))") ➞ ["((())())", "(()(()()))"]

1. Создайте две функции toCamelCase () и toSnakeCase (), каждая из которых берет одну строку и преобразует ее либо в camelCase, либо в snake\_case.

Пример:

toCamelCase("hello\_edabit") ➞ "helloEdabit"

toSnakeCase("helloEdabit") ➞ "hello\_edabit"

toCamelCase("is\_modal\_open") ➞ "isModalOpen"

toSnakeCase("getColor") ➞ "get\_color"

Примечание:

– Snake case — стиль написания составных слов, при котором несколько слов разделяются символом подчеркивания (\_), и не имеют пробелов в записи, причём каждое слово обычно пишется с маленькой буквы — «foo\_bar», «hello\_world» и т. д.

– CamelCase — стиль написания составных слов, при котором несколько слов пишутся слитно без пробелов, при этом каждое слово внутри фразы пишется с прописной буквы. Стиль получил название CamelCase, поскольку прописные буквы внутри слова напоминают горбы верблюда

1. Напишите функцию, которая вычисляет сверхурочную работу и оплату, связанную с сверхурочной работой.

Работа с 9 до 5: обычные часы работы

После 5 вечера это сверхурочная работа

Ваша функция получает массив с 4 значениями:

– Начало рабочего дня, в десятичном формате, (24-часовая дневная нотация)

– Конец рабочего дня. (Тот же формат)

– Почасовая ставка

– Множитель сверхурочных работ

Ваша функция должна возвращать:

$ + заработанные в тот день (округлены до ближайшей сотой)

Пример:

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"

Примечание:

С 16 до 17 регулярно, поэтому 1 \* 30 = 30

С 17 до 18 сверхурочно, поэтому 1 \* 30 \* 1,8 = 54

30 + 54 = 84,00 $

1. Индекс массы тела (ИМТ) определяется путем измерения вашего веса в килограммах и деления на квадрат вашего роста в метрах. Категории ИМТ таковы:

Недостаточный вес: <18,5

Нормальный вес: 18.5-24.9

Избыточный вес: 25 и более

Создайте функцию, которая будет принимать вес и рост (в килограммах, фунтах, метрах или дюймах) и возвращать ИМТ и связанную с ним категорию. Округлите ИМТ до ближайшей десятой.

Пример:

BMI("205 pounds", "73 inches") ➞ "27.0 Overweight"

BMI("55 kilos", "1.65 meters") ➞ "20.2 Normal weight"

BMI("154 pounds", "2 meters") ➞ "17.5 Underweight"

1. Создайте функцию, которая принимает число и возвращает его мультипликативное постоянство, которое представляет собой количество раз, которое вы должны умножать цифры в num, пока не достигнете одной цифры.

Пример:

bugger(39) ➞ 3

// Because 3 \* 9 = 27, 2 \* 7 = 14, 1 \* 4 = 4 and 4 has only one digit.

bugger(999) ➞ 4

// Because 9 \* 9 \* 9 = 729, 7 \* 2 \* 9 = 126, 1 \* 2 \* 6 = 12, and finally 1 \* 2 = 2.

bugger(4) ➞ 0

// Because 4 is already a one-digit number.

1. Напишите функцию, которая преобразует строку в звездную стенографию. Если символ повторяется n раз, преобразуйте его в символ\*n.

Пример:

toStarShorthand("abbccc") ➞ "ab\*2c\*3"

toStarShorthand("77777geff") ➞ "7\*5gef\*2"

toStarShorthand("abc") ➞ "abc"

toStarShorthand("") ➞ ""

1. Создайте функцию, которая возвращает true, если две строки рифмуются, и false в противном случае. Для целей этого упражнения две строки рифмуются, если последнее слово из каждого предложения содержит одни и те же гласные.

Пример:

doesRhyme("Sam I am!", "Green eggs and ham.") ➞ true

doesRhyme("Sam I am!", "Green eggs and HAM.") ➞ true

// Capitalization and punctuation should not matter.

doesRhyme("You are off to the races", "a splendid day.") ➞ false

doesRhyme("and frequently do?", "you gotta move.") ➞ false

Примечание:

– Без учета регистра.

– Здесь мы не обращаем внимания на такие случаи, как "thyme" и "lime".

– Мы также игнорируем такие случаи, как "away" и "today" (которые технически рифмуются, хотя и содержат разные гласные).

1. Создайте функцию, которая принимает два целых числа и возвращает true, если число повторяется три раза подряд в любом месте в num1 и то же самое число повторяется два раза подряд в num2.

Пример:

trouble(451999277, 41177722899) ➞ True

trouble(1222345, 12345) ➞ False

trouble(666789, 12345667) ➞ True

trouble(33789, 12345337) ➞ False

1. Предположим, что пара одинаковых символов служит концами книги для всех символов между ними. Напишите функцию, которая возвращает общее количество уникальных символов (книг, так сказать) между всеми парами концов книги.

Эта функция будет выглядеть следующим образом:

countUniqueBooks("stringSequence", "bookEnd")

Пример:

countUniqueBooks("AZYWABBCATTTA", 'A') ➞ 4

// 1st bookend group: "AZYWA" : 3 unique books: "Z", "Y", "W"

// 2nd bookend group: "ATTTA": 1 unique book: "T"

countUniqueBooks("$AA$BBCATT$C$$B$", '$') ➞ 3

countUniqueBooks("ZZABCDEF", 'Z') ➞ 0

5/6

1. Пришло время отправлять и получать секретные сообщения.

Создайте две функции, которые принимают строку и массив и возвращают закодированное или декодированное сообщение.

Первая буква строки или первый элемент массива представляет собой символьный код этой буквы. Следующие элементы-это различия между символами: например, A +3 --> C или z -1 --> y.

Пример:

encrypt("Hello") ➞ [72, 29, 7, 0, 3]

// H = 72, the difference between the H and e is 29 (upper- and lowercase).

// The difference between the two l's is obviously 0.

decrypt([ 72, 33, -73, 84, -12, -3, 13, -13, -68 ]) ➞ "Hi there!"

encrypt("Sunshine") ➞ [83, 34, -7, 5, -11, 1, 5, -9]

1. Создайте функцию, которая принимает имя шахматной фигуры, ее положение и целевую позицию. Функция должна возвращать true, если фигура может двигаться к цели, и false, если она не может этого сделать.

Возможные входные данные - "пешка", "конь", "слон", "Ладья", "Ферзь"и " король".

Пример:

canMove("Rook", "A8", "H8") ➞ true

canMove("Bishop", "A7", "G1") ➞ true

canMove("Queen", "C4", "D6") ➞ false

1. Входная строка может быть завершена, если можно добавить дополнительные буквы, и никакие буквы не должны быть удалены, чтобы соответствовать слову. Кроме того, порядок букв во входной строке должен быть таким же, как и порядок букв в последнем слове.

Создайте функцию, которая, учитывая входную строку, определяет, может ли слово быть завершено.

Пример:

canComplete("butl", "beautiful") ➞ true

// We can add "ea" between "b" and "u", and "ifu" between "t" and "l".

canComplete("butlz", "beautiful") ➞ false

// "z" does not exist in the word beautiful.

canComplete("tulb", "beautiful") ➞ false

// Although "t", "u", "l" and "b" all exist in "beautiful", they are incorrectly ordered.

canComplete("bbutl", "beautiful") ➞ false

// Too many "b"s, beautiful has only 1.

1. Создайте функцию, которая принимает числа в качестве аргументов, складывает их вместе и возвращает произведение цифр до тех пор, пока ответ не станет длиной всего в 1 цифру.

Пример:

sumDigProd(16, 28) ➞ 6

sumDigProd(0) ➞ 0

sumDigProd(1, 2, 3, 4, 5, 6) ➞ 2

1. Напишите функцию, которая выбирает все слова, имеющие все те же гласные (в любом порядке и / или количестве), что и первое слово, включая первое слово.

Пример:

sameVowelGroup(["toe", "ocelot", "maniac"]) ➞ ["toe", "ocelot"]

sameVowelGroup(["many", "carriage", "emit", "apricot", "animal"]) ➞ ["many"]

sameVowelGroup(["hoops", "chuff", "bot", "bottom"]) ➞ ["hoops", "bot", "bottom"]

1. Создайте функцию, которая принимает число в качестве аргумента и возвращает true, если это число является действительным номером кредитной карты, а в противном случае-false.

Номера кредитных карт должны быть длиной от 14 до 19 цифр и проходить тест Луна, описанный ниже:

– Удалите последнюю цифру (это"контрольная цифра").

– Переверните число.

– Удвойте значение каждой цифры в нечетных позициях. Если удвоенное значение имеет более 1 цифры, сложите цифры вместе (например, 8 x 2 = 16 ➞ 1 + 6 = 7).

– Добавьте все цифры.

– Вычтите последнюю цифру суммы (из шага 4) из 10. Результат должен быть равен контрольной цифре из Шага 1.

Пример:

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, 1, 4, 6, 2, 2]

// Step 4: sum = 58

// Step 5: 10 - 8 = 2 (not equal to 6) ➞ false

validateCard(1234567890123452) ➞ true

// Same as above, but check digit checks out.

1. Напишите функцию, которая принимает положительное целое число от 0 до 999 включительно и возвращает строковое представление этого целого числа, написанное на английском языке.

Пример:

numToEng(0) ➞ "zero"

numToEng(18) ➞ "eighteen"

numToEng(126) ➞ "one hundred twenty six"

numToEng(909) ➞ "nine hundred nine"

Тоже самое нужно сделать и для русского языка.

1. Хеш-алгоритмы легко сделать одним способом, но по существу невозможно сделать наоборот. Например, если вы хешируете что-то простое, например, password123, это даст вам длинный код, уникальный для этого слова или фразы. В идеале, нет способа сделать это в обратном порядке. Вы не можете взять хеш-код и вернуться к слову или фразе, с которых вы начали.

Создайте функцию, которая возвращает безопасный хеш SHA-256 для данной строки. Хеш должен быть отформатирован в виде шестнадцатеричной цифры.

Пример:

getSha256Hash("password123") ➞ "ef92b778bafe771e89245b89ecbc08a44a4e166c06659911881f383d4473e94f"

getSha256Hash("Fluffy@home") ➞ "dcc1ac3a7148a2d9f47b7dbe3d733040c335b2a3d8adc7984e0c483c5b2c1665"

getSha256Hash("Hey dude!") ➞ "14f997f08b8ad032dcb274198684f995d34043f9da00acd904dc72836359ae0f"

Примечание:

Бонус, если вы можете сделать это без импорта каких-либо библиотек ;)

1. Напишите функцию, которая принимает строку и возвращает строку с правильным регистром для заголовков символов в серии "Игра престолов".

Слова and, the, of и in должны быть строчными. Все остальные слова должны иметь первый символ в верхнем регистре, а остальные-в Нижнем.

Пример:

correctTitle("jOn SnoW, kINg IN thE noRth.")

➞ "Jon Snow, King in the North."

correctTitle("sansa stark, lady of winterfell.")

➞ "Sansa Stark, Lady of Winterfell."

correctTitle("TYRION LANNISTER, HAND OF THE QUEEN.")

➞ "Tyrion Lannister, Hand of the Queen."

Примечания:

– Знаки препинания и пробелы должны оставаться в своих первоначальных положениях.

– Дефисные слова считаются отдельными словами.

– Будьте осторожны со словами, которые содержат and, the, of или in.

1. Как указано в онлайн-энциклопедии целочисленных последовательностей:

Гексагональная решетка - это привычная двумерная решетка, в которой каждая точка имеет 6 соседей.

Центрированное шестиугольное число - это центрированное фигурное число, представляющее шестиугольник с точкой в центре и всеми другими точками, окружающими центральную точку в шестиугольной решетке.

Illustration of initial terms:

.

. o o o o

. o o o o o o o o

. o o o o o o o o o o o o

. o o o o o o o o o o o o o o o o

. o o o o o o o o o o o o

. o o o o o o o o

. o o o o

.

. 1 7 19 37

.

Напишите функцию, которая принимает целое число n и возвращает "недопустимое", если n не является центрированным шестиугольным числом или его иллюстрацией в виде многострочной прямоугольной строки в противном случае.

Пример:

hexLattice(1) ➞ " o "

// o

hexLattice(7) ➞ " o o \n o o o \n o o "

// o o

// o o o

// o o

hexLattice(19) ➞ " o o o \n o o o o \n o o o o o \n o o o o \n o o o "

// o o o

// o o o o

// o o o o o

// o o o o

// o o o

hexLattice(21) ➞ "Invalid"

**6/6**

1. Число Белла - это количество способов, которыми массив из n элементов может быть разбит на непустые подмножества. Создайте функцию, которая принимает число n и возвращает соответствующее число Белла.

Пример:

bell(1) ➞ 1

// sampleArr = [1]

// possiblePartitions = [[[1]]]

bell(2) ➞ 2

// sampleArr = [1, 2]

// possiblePartitions = [[[1, 2]], [[1], [2]]]

bell(3) ➞ 5

// sampleArr = [1, 2, 3]

// possiblePartitions = [[[1, 2, 3]], [[1, 2], [3]], [[1], [2, 3]], [[1, 3], [2]], [[1], [2], [3]]]

1. В «поросячей латыни» (свинский латинский) есть два очень простых правила:

– Если слово начинается с согласного, переместите первую букву (буквы) слова до гласного до конца слова и добавьте «ay» в конец.

have ➞ avehay

cram ➞ amcray

take ➞ aketay

cat ➞ atcay

shrimp ➞ impshray

trebuchet ➞ ebuchettray

* Если слово начинается с гласной, добавьте "yay" в конце слова.

ate ➞ ateyay

apple ➞ appleyay

oaken ➞ oakenyay

eagle ➞ eagleyay

Напишите две функции, чтобы сделать переводчик с английского на свинский латинский. Первая функция translateWord (word) получает слово на английском и возвращает это слово, переведенное на латинский язык. Вторая функция translateSentence (предложение) берет английское предложение и возвращает это предложение, переведенное на латинский язык.

Пример:

translateWord("flag") ➞ "agflay"

translateWord("Apple") ➞ "Appleyay"

translateWord("button") ➞ "uttonbay"

translateWord("") ➞ ""

translateSentence("I like to eat honey waffles.") ➞ "Iyay ikelay otay eatyay oneyhay afflesway."

translateSentence("Do you think it is going to rain today?") ➞ "Oday youyay inkthay ityay isyay oinggay otay ainray odaytay?"

Примечание:

– Регулярные выражения помогут вам не исказить пунктуацию в предложении.

– Если исходное слово или предложение начинается с заглавной буквы, перевод должен сохранить свой регистр

1. Учитывая параметры RGB (A) CSS, определите, является ли формат принимаемых значений допустимым или нет. Создайте функцию, которая принимает строку (например, " rgb(0, 0, 0)") и возвращает true, если ее формат правильный, в противном случае возвращает false.

Пример:

validColor("rgb(0,0,0)") ➞ true

validColor("rgb(0,,0)") ➞ false

validColor("rgb(255,256,255)") ➞ false

validColor("rgba(0,0,0,0.123456789)") ➞ true

1. Создайте функцию, которая принимает URL (строку), удаляет дублирующиеся параметры запроса и параметры, указанные во втором аргументе (который будет необязательным массивом).

Пример:

stripUrlParams("https://edabit.com?a=1&b=2&a=2") ➞ "https://edabit.com?a=2&b=2"

stripUrlParams("https://edabit.com?a=1&b=2&a=2", ["b"]) ➞ "https://edabit.com?a=2"

stripUrlParams("https://edabit.com", ["b"]) ➞ "https://edabit.com"

Примечание:

– Второй аргумент paramsToStrip является необязательным.

– paramsToStrip может содержать несколько параметров.

– Если есть повторяющиеся параметры запроса с разными значениями, используйте значение последнего встречающегося параметра (см. Примеры № 1 и № 2 выше).

1. Напишите функцию, которая извлекает три самых длинных слова из заголовка газеты и преобразует их в хэштеги. Если несколько слов одинаковой длины, найдите слово, которое встречается первым.

Пример:

getHashTags("How the Avocado Became the Fruit of the Global Trade")

➞ ["#avocado", "#became", "#global"]

getHashTags("Why You Will Probably Pay More for Your Christmas Tree This Year")

➞ ["#christmas", "#probably", "#will"]

getHashTags("Hey Parents, Surprise, Fruit Juice Is Not Fruit")

➞ ["#surprise", "#parents", "#fruit"]

getHashTags("Visualizing Science")

➞ ["#visualizing", "#science"]

Примечание:

– Если заголовок содержит менее 3 слов, просто расположите слова в заголовке по длине в порядке убывания (см. Пример №4).

– Пунктуация не считается с длиной слова.

1. Последовательность Улама начинается с:

ulam = [1, 2]

Следующее число в последовательности - это наименьшее положительное число, равное сумме двух разных чисел (которые уже есть в последовательности) ровно одним способом. Тривиально, это 3, так как в стартовой последовательности есть только 2 числа.

ulam = [1, 2, 3]

Следующее число 4, которое является суммой 3 + 1. 4 также равно 2 + 2, но это уравнение не учитывается, так как 2 добавления должны быть различны.

ulam = [1, 2, 3, 4]

Следующее число не может быть 5, так как 5 = 1 + 4, но также и 5 = 2 + 3. Должен быть только один способ сделать число Улама из 2 различных добавлений, найденных в последовательности. Следующее число 6 (2 + 4). Есть 2 способа сделать 7 (1 + 6 или 3 + 4), поэтому следующий - 8 (2 + 6). И так далее.

ulam = [1, 2, 3, 4, 6, 8, 11, 13, 16, 18, 26, 28, 36, 38, 47, 48, 53, …]

Создайте функцию, которая принимает число n и возвращает n-е число в последовательности Улама.

Пример:

ulam(4) ➞ 4

ulam(9) ➞ 16

ulam(206) ➞ 1856

1. Напишите функцию, которая возвращает самую длинную неповторяющуюся подстроку для строкового ввода.

Пример:

longestNonrepeatingSubstring("abcabcbb") ➞ "abc"

longestNonrepeatingSubstring("aaaaaa") ➞ "a"

longestNonrepeatingSubstring("abcde") ➞ "abcde"

longestNonrepeatingSubstring("abcda") ➞ "abcd"

Примечание:

– Если несколько подстрок связаны по длине, верните ту, которая возникает первой.

– Бонус: можете ли вы решить эту проблему в линейном времени?

1. Создайте функцию, которая принимает арабское число и преобразует его в римское число.

Пример:

convertToRoman(2) ➞ "II"

convertToRoman(12) ➞ "XII"

convertToRoman(16) ➞ "XVI"

Примечание:

– Все римские цифры должны быть возвращены в верхнем регистре.

– Самое большое число, которое может быть представлено в этой нотации, - 3,999.

1. Создайте функцию, которая принимает строку и возвращает true или false в зависимости от того, является ли формула правильной или нет.

Пример:

formula("6 \* 4 = 24") ➞ true

formula("18 / 17 = 2") ➞ false

formula("16 \* 10 = 160 = 14 + 120") ➞ false

1. Число может не быть палиндромом, но его потомком может быть. Прямой потомок числа создается путем суммирования каждой пары соседних цифр, чтобы создать цифры следующего числа.

Например, 123312 – это не палиндром, а его следующий потомок 363, где: 3 = 1 + 2; 6 = 3 + 3; 3 = 1 + 2.

Создайте функцию, которая возвращает значение true, если само число является палиндромом или любой из его потомков вплоть до 2 цифр (однозначное число - тривиально палиндром).

Пример:

palindromedescendant(11211230) ➞ true

// 11211230 ➞ 2333 ➞ 56 ➞ 11

palindromeDescendant(13001120) ➞ true

// 13001120 ➞ 4022 ➞ 44

palindromeDescendant(23336014) ➞ true

// 23336014 ➞ 5665

palindromeDescendant(11) ➞ true

// Number itself is a palindrome.

Примечание:

– Числа всегда будут иметь четное число цифр.