## General Notes:

Most of the exercises should take less than 30-45 minutes, If you are stuck, move to the next one, solve as many as you can, and get back later to the exercises you were stuck at.

# Basic

## Ex1.1 - Yes or No

Complete the method that takes a boolean value and return a "Yes" string for true, or a "No" string for false.

# Simple Math

## Ex2.1 - Sum of lowest numbers

Create a function that returns the sum of the two lowest positive numbers given an array of minimum 4 positive integers. No floats or non-positive integers will be passed.
For example, when an array is passed like [19, 5, 42, 2, 77], the output should be 7.
[10, 343445353, 3453445, 3453545353453] should return 3453455.

## Ex2.2 - One and Zero - Binary

Given an array of ones and zeroes, convert the equivalent binary value to an integer.

Eg: [0, 0, 0, 1] is treated as 0001 which is the binary representation of 1.

Examples:

Testing: [0, 0, 0, 1] ==> 1
Testing: [0, 0, 1, 0] ==> 2
Testing: [0, 1, 0, 1] ==> 5
Testing: [1, 0, 0, 1] ==> 9
Testing: [0, 0, 1, 0] ==> 2
Testing: [0, 1, 1, 0] ==> 6
Testing: [1, 1, 1, 1] ==> 15
Testing: [1, 0, 1, 1] ==> 11
However, the arrays can have varying lengths, not just limited to 4.

## Ex2.3 - Find the Next Perfect Square

You might know some pretty large perfect squares. But what about the NEXT one?

Complete the findNextSquare method that finds the next integral perfect square after the one passed as a parameter. Recall that an integral perfect square is an integer n such that sqrt(n) is also an integer.

If the parameter is itself not a perfect square then -1 should be returned. You may assume the parameter is positive.

Examples:

findNextSquare(121) --> returns 144
findNextSquare(625) --> returns 676
findNextSquare(114) --> returns -1 since 114 is not a perfect

## Ex2.4 - Unique

There is an array with some numbers. All numbers are equal except for one. Try to find it!
findUniq([ 1, 1, 1, 2, 1, 1 ]) === 2
findUniq([ 0, 0, 0.55, 0, 0 ]) === 0.55
It’s guaranteed that array contains at least 3 numbers.

## Ex2.5 - Summation

Write a program that finds the summation of every number from 1 to num. The number will always be a positive integer greater than 0.
For example:
summation(2) -> 3
1 + 2

summation(8) -> 36
1 + 2 + 3 + 4 + 5 + 6 + 7 + 8

## Ex2.6 - Years and Centuries

The first century spans from the year 1 up to and including the year 100, The second - from the year 101 up to and including the year 200, etc.
Task :
Given a year, return the century it is in.
Input , Output Examples ::
centuryFromYear(1705)  returns (18)
centuryFromYear(1900)  returns (19)
centuryFromYear(1601)  returns (17)
centuryFromYear(2000)  returns (20)

## Ex2.7 - Basic Math

Your task is to create a function that does four basic .
The function should take three arguments - operation(string/char), value1(number), value2(number).
The function should return result of numbers after applying the chosen operation.
Examples
basicOp('+', 4, 7)         // Output: 11
basicOp('-', 15, 18)       // Output: -3
basicOp('*', 5, 5)         // Output: 25
basicOp('/', 49, 7)        // Output: 7

# Math In Story

## Ex3.1 - Growth Of population

In a small town the population is p0 = 1000 at the beginning of a year. The population regularly increases by 2 percent per year and moreover 50 new inhabitants per year come to live in the town. How many years does the town need to see its population greater or equal to p = 1200 inhabitants?

At the end of the first year there will be:
1000 + 1000 * 0.02 + 50 => 1070 inhabitants

At the end of the 2nd year there will be:
1070 + 1070 * 0.02 + 50 => 1141 inhabitants (number of inhabitants is an integer)

At the end of the 3rd year there will be:
1141 + 1141 * 0.02 + 50 => 1213

It will need 3 entire years.

More generally given parameters:

- p0, percent, aug (inhabitants coming or leaving each year), p (population to surpass)
- the function nb_year should return n number of entire years needed to get a population greater or equal to p.
- aug is an integer, percent a positive or null number, p0 and p are positive integers (> 0)

Examples:

```
nb_year(1500, 5, 100, 5000) -> 15
nb_year(1500000, 2.5, 10000, 2000000) -> 10
```

Note: Don't forget to convert the percent parameter as a percentage in the body of your function: if the parameter percent is 2 you have to convert it to 0.02.

# Ex3.2 - People on the Bus

Number of people in the bus
There is a bus moving in the city, and it takes and drop some people in each bus stop.
You are provided with a list (or array) of integer arrays (or tuples). Each integer array has two items which represent number of people get into bus (The first item) and number of people get off the bus (The second item) in a bus stop.
Your task is to return number of people who are still in the bus after the last bus station (after the last array). Even though it is the last bus stop, the bus is not empty and some people are still in the bus, and they are probably sleeping there :D
Take a look on the test cases.
Please keep in mind that the test cases ensure that the number of people in the bus is always >= 0. So the return integer can't be negative.
The second value in the first integer array is 0, since the bus is empty in the first bus stop.

# Advanced Math

## Ex4.1 - Fibonacci -

“Write a function to return an n element in Fibonacci sequence” is one of the most common questions you can hear during the coding challenge interview part. In this blogpost I’m going to walk through the two of the most typical solutions for this problem and also cover a dreadful (for most of novice developers) topic of time complexity.
So what is a Fibonacci sequence? According to Wikipedia:
“In mathematics, the Fibonacci numbers are the numbers in the following integer sequence, called the Fibonacci sequence, and characterized by the fact that every number after the first two is the sum of the two preceding ones.”
Depending on the chosen starting point of the sequence (0 or 1) the sequence would look like this:
1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …
or this:
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …

## Ex4.2 - Tribonacci -

Well met with Fibonacci bigger brother, AKA Tribonacci.
As the name may already reveal, it works basically like a Fibonacci, but summing the last 3 (instead of 2) numbers of the sequence to generate the next. And, worse part of it, regrettably I won't get to hear non-native Italian speakers trying to pronounce it :(
So, if we are to start our Tribonacci sequence with [1, 1, 1] as a starting input (AKA signature), we have this sequence:
[1, 1 ,1, 3, 5, 9, 17, 31, ...]
But what if we started with [0, 0, 1] as a signature? As starting with [0, 1] instead of [1, 1] basically shifts the common Fibonacci sequence by once place, you may be tempted to think that we would get the same sequence shifted by 2 places, but that is not the case and we would get:
[0, 0, 1, 1, 2, 4, 7, 13, 24, ...]
Well, you may have guessed it by now, but to be clear: you need to create a fibonacci function that given a signature array/list, returns the first n elements - signature included of the so seeded sequence.
Signature will always contain 3 numbers; n will always be a non-negative number; if n == 0, then return an empty array (except in C return NULL) and be ready for anything else which is not clearly specified ;)

# Basic Iteration Logic

## Ex5.1 - trimming string

It's pretty straightforward. Your goal is to create a function that removes the first and last characters of a string. You're given one parameter, the original string. You don't have to worry with strings with less than two characters.

## Ex5.2 - String Repeat

Write a function called repeat_str which repeats the given string src exactly count times.
repeatStr(6, "I") // "IIIIII"
repeatStr(5, "Hello") // "HelloHelloHelloHelloHello"

## Ex5.3 - To Camel Case

Complete the method/function so that it converts dash/underscore delimited words into camel casing. The first word within the output should be capitalized only if the original word was capitalized (known as Upper Camel Case, also often referred to as Pascal case).

Examples
toCamelCase("the-stealth-warrior") // returns "theStealthWarrior"

toCamelCase("The_Stealth_Warrior") // returns "TheStealthWarrior"

## Ex5.4 - To Weird Case

Write a function toWeirdCase (weirdcase in Ruby) that accepts a string, and returns the same string with all even indexed characters in each word upper cased, and all odd indexed characters in each word lower cased. The indexing just explained is zero based, so the zero-ith index is even, therefore that character should be upper cased.
The passed in string will only consist of alphabetical characters and spaces(' '). Spaces will only be present if there are multiple words. Words will be separated by a single space(' ').
Examples:
toWeirdCase( "String" );//=> returns "StRiNg"
toWeirdCase( "Weird string case" );//=> returns "WeIrD StRiNg CaSe"

## Ex5.5 - Abbreviate two words

Write a function to convert a name into initials. This kata strictly takes two words with one space in between them.
The output should be two capital letters with a dot separating them.
It should look like this:
Sam Harris => S.H
Patrick Feeney => P.F

## Ex5.6 - Mask

Usually when you buy something, you're asked whether your credit card number, phone number or answer to your most secret question is still correct. However, since someone could look over your shoulder, you don't want that shown on your screen. Instead, we mask it.
Your task is to write a function maskify, which changes all but the last four characters into '#'.
Examples
maskify("4556364607935616") == "############5616"
maskify(     "64607935616") ==      "#######5616"
maskify(              "1") ==              "1"
maskify(               "") ==              ""

// "What was the name of your first pet?"
maskify("Skippy")                                   == "##ippy"
maskify("Nananananananananananananananananana Batman!") ==
"####################################man!"

## Ex5.7 - shortest words

Simple, given a string of words, return the length of the shortest word(s).
String will never be empty and you do not need to account for different data types.

# Advanced Iteration Logic

## Ex6.1 - Mumbling

This time no story, no theory. The examples below show you how to write function `accum`:
Examples:
accum("abcd") -> "A-Bb-Ccc-Dddd"
accum("RqaEzty") -> "R-Qq-Aaa-Eeee-Zzzzz-Ttttt-Yyyyyyy"
accum("cwAt") -> "C-Ww-Aaa-Tttt"
The parameter of accum is a string which includes only letters from `a..z` and `A..Z`.

## Ex6.2 - Counting Duplicates

Count the number of Duplicates
Write a function that will return the count of distinct case-insensitive alphabetic characters and numeric digits that occur more than once in the input string. The input string can be assumed to contain only alphabets (both uppercase and lowercase) and numeric digits.
Example
"abcde" -> 0 # no characters repeats more than once
"aabbcde" -> 2 # 'a' and 'b'
"aabBcde" -> 2 # 'a' occurs twice and 'b' twice (`b` and `B`)
"indivisibility" -> 1 # 'i' occurs six times
"Indivisibilities" -> 2 # 'i' occurs seven times and 's' occurs twice
"aA11" -> 2 # 'a' and '1'
"ABBA" -> 2 # 'A' and 'B' each occur twice

## Ex6.3 - organize strings

Take 2 strings s1 and s2 including only letters from ato z. Return a new sorted string, the longest possible, containing distinct letters,

each taken only once - coming from s1 or s2.

Examples:
a = "xyaabbbccccdefww"
b = "xxxxyyyyabklmopq"
longest(a, b) -> "abcdefklmopqwxy"

a = "abcdefghijklmnopqrstuvwxyz"
longest(a, a) -> "abcdefghijklmnopqrstuvwxyz"

# Ex6.4 - isogram

An isogram is a word that has no repeating letters, consecutive or non-consecutive. Implement a function that determines whether a string that contains only letters is an isogram. Assume the empty string is an isogram. Ignore letter case.
isIsogram("Dermatoglyphics") == true
isIsogram("aba") == false
isIsogram("moOse") == false // -- ignore letter case

# Implement Functionality

## Ex7 - Implement The Following JS Methods -

Implement the following methods -

- Filter
- ForEach
- Map

Using only for(), array and objects (without other js methods)

# Find Bugs \ Debug Code

Bug !!!!!