# CHALLENGES

# ISBN Validator:

There are two ISBN standards: ISBN-10 and ISBN-13. Support for ISBN-13 is essential, whereas support for ISBN-10 is optional. Here are some valid examples of each:

| Format | Examples |
|---|---|
| ISBN-10 | 0471958697 |
| | 0 471 60695 2 |
| | 0-470-84525-2 |
| | 0-321-14653-0 |
| | |
| ISBN-13 | 9780470059029 |
| | 978 0 471 48648 0 |
| | 978-0596809485 |
| | 978-0-13-149505-0 |
| | 978-0-262-13472-9 |

ISBN-10 is made up of 9 digits plus a check digit (which may be 'X') and ISBN-13 is made up of 12 digits plus a check digit. Spaces and hyphens may be included in a code, but are not significant. This means that 9780471486480 is equivalent to 978-0-471-48648-0 and 978 0 471 48648 0.

The check digit for ISBN-10 is calculated by multiplying each digit by its position (i.e., 1 x 1st digit, 2 x 2nd digit, etc.), summing these products together, doing a modulo of 11 and then taking that result and subtracting it from modulo 11 (with 'X' being used if the result is 10).

The check digit for ISBN-13 is calculated by multiplying each digit alternately by 1 or 3 (i.e., 1 x 1st digit, 3 x 2nd digit, 1 x 3rd digit, 3 x 4th digit, etc.), summing these products together, taking modulo 10 of the result and subtracting this value from 10, and then taking the modulo 10 of the result again to produce a single digit.

**Create a function that takes a string and returns true if that is a valid ISBN-13 and false otherwise.**

**Create a function that takes a string and returns true if that is a valid ISBN-10 and false otherwise.**

```
function isValidIsbn10(isbnNumber) {

 // code ..

}

function isValidIsbn13(isbnNumber) {

// code ..

}
```

**String Expression:**

Write a function such that when given a string parameter, which will perform a mathematical operation and return the result, in words.

Numbers 0-9 and the symbols "plus" and "minus" are the only inputs allowed in the single parameter the program receives.

For example:

```
stringExpression( "onezeropluseight" ) ===> "oneeight"
stringExpression( "oneminusoneone" ) ===> "negativeonezero"
```

## Scrambler

Given two strings, when passed a scrambled string, if passed a second one the method will return true if the second word contains all letters in the rest.

For example:

scrambler(“cdoer”, “coder”) ===> true

scrambler(“hello”, “hkllo”) ===> false

## Human Readable Time

Write a function, which takes a non-negative integer (seconds) as input and returns the time in a human readable format (HH:MM:SS)

HH = hours, padded to 2 digits, range 00-99

MM = hours, padded to 2 digits, range 00-59

SS = seconds, padded to 2 digits, range 00-59

The maximum time never exceeds 359999 (99:59:59)

**Row Weights**

Several people are standing in a row divided into two teams.

The first person goes into team 1, the second into team 2, the third into team 1 and so on.

Challenge: Given an array of positive numbers (the weights of the people), return a new array/tuple of two numbers, where the first one is the total weight of team 1 and the second one is the total weight of team 2

Array sizes provided must be at least 1.

For example: rowWeights([13, 27, 49]) ==> returns (62, 27)

as team 1 contains 13 and 49 = 62 and team 2 only contains 27, so 27 is returned.

# Blue Ticket:

You have a blue lottery ticket with numbers A, B and C on it. This makes three pairs, which we'll say are AB, BC and AC.

Consider the sum of the numbers in each pair. If any pair sums to exactly 10, the result returned is 10. Otherwise if AB sum is exactly 10 more than either BC or AC sums, then the result is 5. Otherwise the result is 0.

Examples:

blueTicket(9, 1, 0) ===> 10

blueTicket(9, 2, 0) ===> 0

blueTicket(6, 1, 4) ===> 10

```
function blueTicket(a, b, c) {

// code ..

}
```

## Blackjack

Given two numbers greater than 0, return whichever is nearest to 21 without going over. Return 0 if they both go over 21.

blackJack(19, 21) ===> 21

blackJack(21, 19) ===> 21

blackJack(19, 22) ===> 19

blackJack(99, 23) ===> 0

# Lone Sum

Given 3 numbers, a, b and c, return their sum.

However, if one of the values is the same as another of the values, it does not count towards the sum.

For example:

loneSum(1, 2, 3) ===> 6

loneSum(3, 2, 3) ===> 2

loneSum(1, 1, 1) ===> 0

# FindLowestIndex

Return the index of the lowest value in an array

The input array will have at least one element in it.

findLowestIndexInArr([99, 98, 97, 96, 95]) ===> 4

**SortByHeight**

Some people are standing in a row in a park. There are trees between them which cannot be moved. Your task is to rearrange the people by their height in ascending numeric order without moving the trees.

Trees in the array are marked as ‘-1’

e.g. [-1, 150, 190, 170, -1, -1, 160, 180] => [-1, 150, 160, 170, -1, -1, 180, 190]