Weekend Project: Credit Card Validator

You're a junior developer working for a software company. The company suspects that users on the web platform have been submitting cards that have invalid numbers. The company would like to know which card issuer with the invalid numbers. You have been assigned this task to check if credit cards are valid.

At the moment the company manually checks credit cards using pencil and paper, but you'll be optimizing the verification process using your knowledge of functions and loops to handle multiple credit cards at a time. Unlike the other manual method.

As you progress through the steps, use the `console.log()` statements to check the output of your loops and functions.

Look over the code in the `app.js`. There are 15 variables that each contain the numbers of separate credit card numbers initialized as strings. They have all been named to reflect their status, i.e. variables that start with `valid` contain a valid number, whereas `invalid` do not, and `notCertain` variables can be either.

There is also a `batch` array that stores all of the provided credit cards in a single array.

1. Create a function, `transformStrArr()`that has a parameter of string that contains the digits of separate credit card numbers (`valid`, `invalid` and `notCertain`). This function will transform card numbers (`valid`, `invalid` and `notCertain`) from string into arrays containing the numeric digits of the card numbers. For example **"1234 2345 4353 3456"** becomes **[1,2,3,4,2,3,4,5,4,3,5,3,4,5,6]**

2. Create another function, `validateCard()` that has a parameter of an array. The purpose of `validateCard()` is to return `true` when an array contains digits of a valid credit card number and `false` when it is invalid. This function should NOT change the values of the original array.

To find out if a credit card number is valid or not, use the Luhn algorithm. Generally speaking, an algorithm is a series of steps that solve a problem — the Luhn algorithm is a series of mathematical calculations used to validate certain identification numbers, e.g. credit card numbers. The calculations in the Luhn algorithm can be broken down as the following steps:

1. Starting from the farthest digit to the right, AKA the check digit, iterate to the left.
2. As you iterate to the left, every other digit is doubled (the check digit is not doubled). If the number is greater than `9` after doubling, subtract `9` from its value.
3. Sum up all the digits in the credit card number.
4. If the sum modulo `10` is `0` (if the sum divided by `10` has a remainder of `0`) then the number is valid, otherwise, it's invalid.

Take a look at an image in the project folder for a visual that outlines the steps.

Check your function using both the provided valid and invalid numbers.

3. Create another function, **`findInvalidCards()`** that has one parameter for a nested array of credit card numbers. The role of **`findInvalidCards()`** is to check through the nested array for which numbers are invalid, and return another nested array of invalid cards.

4. After finding all the invalid credit card numbers, it's also necessary to identify the credit card companies that have possibly issued these faulty numbers. Create a function, **`idInvalidCardIssuers()`** that has one parameter for a nested array of invalid numbers and returns an array of companies.

Currently, there are 4 accepted card issuers which each have unique first digits. The following table shows which digit is unique to which card issuer:

First Digit	Card Issuer
3	Amex (American Express)
4	Visa
5	Mastercard
6	Discover

If the number doesn't start with any of the numbers listed, print out a message like: "Card Issuer not found".

`idInvalidCardIssuers()` should return an array of companies that have mailed out cards with invalid numbers. This array should NOT contain duplicates, i.e. even if there are two invalid Visa cards, `"Visa"` should only appear once in the array.