Project Goals

Context: The company that you work for suspects that credit card distributors have been mailing out cards that have invalid numbers. In this project, you have the role of a clerk who checks if credit cards are valid. Every other clerk currently checks 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 clerks, you can spend the rest of your time relaxing!

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

Prerequisites

1.

In order to complete this project, you should have completed the first few sections of Introduction to JavaScript (through Learn JavaScript: Loops).

Project Requirements

2.

Look over the starter code. There are 15 arrays that each contain the digits of separate credit card numbers. They all have prefixes to reflect their status, i.e. variables that start with valid contain a valid number, whereas invalid do not, and mystery variables can be either. There is also a batch array that stores all of the provided credit cards in a single array.

You'll use these arrays later to check if your functions are working properly.

3.

Create a function, validateCred() that has a parameter of an array. The purpose of validateCred() 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 mutate 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.

Here's a visual that outlines the steps. Check your function using both the provided valid and invalid numbers.

4.

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.

5.

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, idInvalidCardCompanies() that has one parameter for a nested array of invalid numbers and returns an array of companies.

Currently, there 4 accepted companies which each have unique first digits. The following table shows which digit is unique to which company:

First Digit	**Company**
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: “Company not found”.

idInvalidCardCompanies() 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.