Computational Thinking

Group Project:

Credit Cards

Vrije Universiteit Amsterdam

Project Title: Credit cards

The following problem is modelled after a real-life task.

We will assume the following in this problem:

Usually, credit card numbers follow a certain pattern. A credit card number is between 13 and 16 digits. The number starts with:

- 4 for Visa cards
- 5 for Master cards
- 37 for American Express cards
- 6 for Discover cards

In 1954, Hans Luhn, while working for IBM proposed an algorithm for validating credit card numbers. The algorithm is useful in determining whether a card number is entered correctly.

Credit card numbers are generated following this validity check. This is also known as the *Luhn check* or the *Mod 10 check*. It can be formulated as follows

For example, take the card number 4388576018402626:

Steps:

1. Double every second digit from right to left. If doubling of a digit results in a two-digit number, add up the two digits to get a single-digit number.

2 * 2 = 4
2 * 2 = 4
4 * 2 = 8
1 * 2 = 2
6 * 2 = 12 (1 + 2 = 3)
5 * 2 = 10 (1 + 0 = 1)
8 * 2 = 16 (1 + 6 = 7)
4 * 2 = 8

2. Now add all single-digit numbers from Step 1.
4 + 4 + 8 + 2 + 3 + 1 + 7 + 8 = 37

3. Add all digits in the odd places from right to left in the card number.
6 + 6 + 0 + 8 + 0 + 7 + 8 + 3 = 38

4. Sum the results from Step 2 and Step 3.
37 + 38 = 75

5. If the result in Step 4 is divisible by **10**, the card number is correct. Otherwise, the number is invalid. For instance, the sequence 4388576018402626 is not a valid credit card number, but the number 4388576018410707 is valid.

Simulation
John wants to simulate a program that can detect whether a provided number is a valid credit card number, following the rules described above. He is allowed to approach this problem creatively, making his own assumptions of what is allowed (for example, are we allowed to provide shorter numbers? They will be invalid anyway. How would his algorithm account for that? Handling such cases can be introduced in a second, more sophisticated version of his first algorithm.

You may assume the computer will be generating numbers randomly OR that a user will be entering a number of a few digits.

Your task:

Familiarize yourself with the idea of using an **algorithm, pseudocode, and Python** before you start with this project.

1. Create an algorithm that allows the simulation to exist. You can choose to do this for the base or the more sophisticated version. (35 pts)
2. Translate your algorithm to a flowchart. (30 pts)
3. Implement your algorithm in Python. (35)