Let's Go Shopping

THE TASK

In this task you need to take a user's credit card input and test if it is a valid credit card number.

Design a simple form to allow the user to input a credit card number (some test numbers are provided below).

Test if the credit card number is valid.

Output an appropriate message, just showing the last 4 digits.

e.g.

Your credit card number ending in 1234 is valid

Your credit card number ending in 1234 is invalid

Replace 1234 with the last 4 digits of their input.

HOW TO TEST IF A CREDIT CARD NUMBER IS VALID

Step 1: Take the credit card number from the user input

4417123456789113

Step 2: Reverse the credit card number

3119876543217144

Step 3: Take each of the odd[#1] position digits and add them

3119876543217144

3 + 1 + 8 + 6 + 4 + 2 + 7 + 4 = **35**

NOTE #2: Think carefully about the logic for this step as it is 'not required' in the traditional sense of it being a separate step. Think about how it relates to Step 5.

Step 4: Take each of the even[#1] digits and multiply by 2

3119876543217144

1*2=2 9*2=18 7*2=14 5*2=10 3*2=6 1*2=2 1*2=2 4*2=8

If the resulting answer for each digit is greater than 9 then subtract 9 from the answer and then add the numbers together

2 + 9 + 5 + 1 + 6 + 2 + 2 + 8 = **35**

Step 5: Add the totals from Step 3 and Step 4

35 + 35 = 70

Step 6: Take the total from Step 5 and check if it is divisible by 10 (with no remainder). If it is then the card number is valid.

SAMPLE CREDIT CARD NUMBERS

4417123456789113	valid
30569309025904	valid
5019817010103742	invalid
412345672124357	invalid

FUNCTIONS YOU MAY FIND USEFUL

Below you will find 4 functions that may help you with this challenge. You will need to visit php.net to find out how to use these

strlen()	find the length of a string
substr()	find part of a string (useful for the last 4 digits!)
str_split()	take the string and convert to an array. You can then use some array functions to work with each digit in turn
array_reverse()	reverses an array

IMPORTANT NOTES

NOTE #1: if you are using an array function to help with the above then remember that an array starts at position zero. So, you need to ensure you understand the difference between the customer supplied rules above (and their positions) as opposed to your solution and where the digit positions will be.

i.e. In the supplied rules above we have:

DIGIT POSITIONS (based on given rules)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
DIGIT VALUE	3	1	1	9	8	7	6	5	4	3	2	1	7	1	4	4

When you create an array you need to remember that the positions start at zero:

DIGIT POSITIONS (based on array positions)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
DIGIT VALUE	3	1	1	9	8	7	6	5	4	3	2	1	7	1	4	4

NOTE #2: In my provided solution I spent some time thinking about the logic and how best to implement the provided 'steps'. This resulted in me coming up with my own step order sequence which still follows the rules but executes them in a cleaner way. You may need to spend some time thinking about the logic!!