# Card Number Verification (Luhn Algorithm)

## Python code

def verify\_card\_number(card\_number):  
 sum\_of\_odd\_digits = 0  
 card\_number\_reversed = card\_number[::-1]  
 odd\_digits = card\_number\_reversed[::2]  
  
  
 for digit in odd\_digits:  
 sum\_of\_odd\_digits += int(digit)  
 sum\_of\_even\_digits = 0  
 even\_digits = card\_number\_reversed[1::2]  
  
  
 for digit in even\_digits:  
 number = int(digit) \* 2  
 if number >= 10:  
 number = (number // 10) + (number % 10)  
 sum\_of\_even\_digits += number  
 total = sum\_of\_odd\_digits + sum\_of\_even\_digits  
 return total % 10 == 0  
  
  
def main():  
 card\_number = '2205-8580-0038-6003'  
 cleaned = card\_number.replace('-', '').replace(' ', '')  
 if verify\_card\_number(cleaned):  
 print('VALID!')  
 else:  
 print('INVALID!')  
main()

## Step-by-step explanation

### Purpose

This script checks whether a provided card number is valid using the Luhn algorithm (a common checksum used to validate credit/debit card numbers).

### 1. Function definition

The function `verify\_card\_number(card\_number)` takes a string `card\_number` consisting only of digits (no spaces or hyphens). It returns True if the number passes the Luhn check, otherwise False.

### 2. Initialize sum for odd-positioned digits

`sum\_of\_odd\_digits = 0` prepares an accumulator for the digits that will be added directly (positions 1, 3, 5, ... counting from the right, 1-indexed).

### 3. Reverse the card number

`card\_number\_reversed = card\_number[::-1]` reverses the string so we can process positions from right to left using simple slice steps. Reversing makes index 0 the rightmost digit (the check digit).

### 4. Extract odd-positioned digits (from the right)

`odd\_digits = card\_number\_reversed[::2]` takes every second character starting at index 0 — these are the digits that are not doubled in the Luhn algorithm.

### 5. Sum odd-positioned digits

The loop `for digit in odd\_digits: sum\_of\_odd\_digits += int(digit)` converts each character to an integer and adds it to `sum\_of\_odd\_digits`.

### 6. Prepare sum for even-positioned digits

`sum\_of\_even\_digits = 0` initializes the accumulator for the digits that must be doubled (positions 2, 4, 6, ... counting from the right).

### 7. Extract even-positioned digits (from the right)

`even\_digits = card\_number\_reversed[1::2]` takes every second character starting at index 1 — these digits will be doubled per Luhn rules.

### 8. Double each even-positioned digit and sum its digits

For each digit in `even\_digits`, the code: `number = int(digit) \* 2`. If `number` is 10 or more, it reduces it to the sum of its digits via `number = (number // 10) + (number % 10)`. Example: 6 \* 2 = 12 → 1 + 2 = 3. This effectively computes the digit sum of the doubled value. The result is added to `sum\_of\_even\_digits`.

### Why we sum digits of the doubled value

Doubling can produce a two-digit number (10–18). Luhn requires adding the individual digits of these products. Computing `(number // 10) + (number % 10)` is a simple way to split and add tens and units digits. Another common trick is to subtract 9 when the doubled value >= 10 (e.g., 12 → 12 - 9 = 3) — both methods are equivalent for this range.

### 9. Compute total and final check

`total = sum\_of\_odd\_digits + sum\_of\_even\_digits`. The card number passes the Luhn check if `total % 10 == 0` (i.e., the total ends with 0). The function returns that boolean expression.

### 10. main() function - input

In `main()` the `card\_number` variable is set to the sample string `'2205-8580-0038-6003'`.

### 11. Cleaning the input

`cleaned = card\_number.replace('-', '').replace(' ', '')` removes hyphens and spaces so that the string contains only digits before passing to the verifier. This is important because `verify\_card\_number` expects digits only.

### 12. Verification and output

The script calls `verify\_card\_number(cleaned)`. If True, it prints `VALID!`; otherwise it prints `INVALID!`. The call `main()` runs the routine when the file is executed.

### 13. Example walkthrough (brief)

Using the example number `2205858000386003` (hyphens removed), the script computes the Luhn checksum step-by-step: reverse, split, double even positions, sum, and check `total % 10`. The script then prints whether the number is valid.

### 14. Edge cases & improvements

• Input validation: currently the code assumes the cleaned string contains only digits. Consider checking `cleaned.isdigit()` and handling empty strings.   
• Alternative digit-sum trick: instead of splitting tens and units, you can subtract 9 when doubled >= 10 (e.g., `number = int(digit) \* 2; if number > 9: number -= 9`).   
• Accept other separators: you might want to strip `.` or other separators.   
• Return types: you could raise an exception on invalid input rather than returning False.   
• Unit tests: write tests for known valid/invalid card numbers.

### 15. Security & usage note

The Luhn algorithm only detects simple transcription errors; it does NOT guarantee the card is genuine or active. Always handle card numbers securely (don’t log or store raw card numbers unless you follow PCI-DSS requirements).

## Output when running the provided code

Sample card number used: 2205-8580-0038-6003

Cleaned digits: 2205858000386003

Result printed by the script: VALID!