Module 3: Luhn's Algorithm

JHU EP 605.206 - Introduction to Programming Using Python

Introduction

In this assignment you will implement Luhn's Algorithm, a checksum for ensuring the validity of identification numbers such as credit card numbers, social security numbers, IMEI's from your mobile phone, and more. Luhn's algorithm cannot detect all errors in an identifier, but it's pretty good at detecting single-digit swap errors, e.g. a user enters 05 instead of 50 as the last two digits of their credit card number while checking out with Amazon Prime. You will use this algorithm to detect whether or not a set of identifiers are correct and to take the first N-1 digits of an identifier, where N is the total number of digits, and generate the final digit to create a valid identification number.

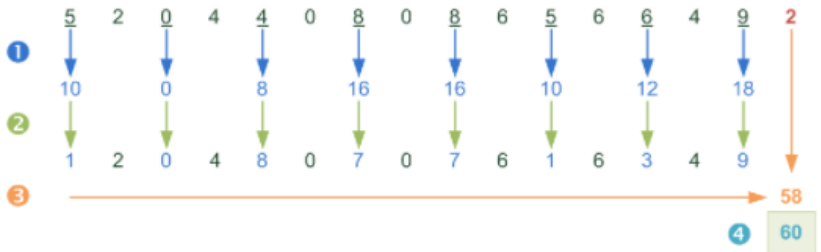
Skills: control structure, logic statements/operators, converting data types, string methods, and more

Luhn's Algorithm

To verify an identifier is correct:

1. Create a variable to keep track of a running total that is initially 0
2. For every other digit beginning with the first (first, third, fifth, and so on), multiply it by 2, then…
    a. If the result is a 1-digit number, add it to the running total
    b. If the result is a 2-digit number, add the digits together, then add that number to the running total
3. For all of the digits in-between, just add them directly to the running total
4. Repeat this process until you reach the end (working left-to-right)
5. If the final running total is evenly divisible by 10, the identifier is valid

Here is an example of a valid identifier:



To generate the correct check digit for an identifier:

1. Create a variable to keep track of a running total that is initially 0
2. For every other digit beginning with the last (N, N-2, N-4, and so on), multiply it by 2 then…
    a. If the result is a 1-digit number, add it to the running total
    b. If the result is a 2-digit number, add the digits together, then add that number to the running total
3. For all of the digits in-between, just add them directly to the running total
4. Repeat this process until you reach the beginning (working right-to-left)
5. Compute (10 – the final running total modulo 10) and the result will become the check digit (the last digit)

Here is an example of how to compute the check digit for the identifier 7992739871:

> Assume an example of an account number "7992739871" (just the "payload", check digit not yet included):

	7	9	9	2	7	3	9	8	7	1
Multipliers	1	2	1	2	1	2	1	2	1	2
	=	=	=	=	=	=	=	=	=	=
	7	**18**	9	**4**	7	**6**	9	**16**	7	**2**
Sum digits	7	**9** (1+8)	9	**4**	7	**6**	9	**7** (1+6)	7	**2**

> The sum of the resulting digits is 67.
>
> The check digit is equal to (10 - (sum modulus 10)), so (10 - (67 modulo 10)) = 3.
>
> This makes the full account number read 79927398713.

Programming Assignment

Part 1 – Luhn's Verification Algorithm

Create a Python program in a file named ***luhn_verify.py*** that:

1. Implements Luhn's verification algorithm above
2. Create a loop that reads two inputs from the user for each of the following `str` (not `int`) inputs:
    a. **3379513561108795**
    b. **4614803459600172**
3. Tells whether each credit card number was valid or invalid using the format provided in the sample outputs below
4. Take a screenshot of each result to be submitted later

Sample Output for Part 1:

```
Checksum = 0
3379513561108795 is a valid CC number.
Checksum = 8
4614803459600172 is an invalid CC number.
```

Part 2 – Luhn's Checksum Generation Algorithm

Create a Python program in a file named ***luhn_generate.py*** that:

1. Implements Luhn's check digit generation algorithm above
2. Generates the correct check digit (16^{th} digit) for the invalid credit card number above
    a. **461480345960017**
3. Displays the full, valid credit card number along with the newly computed check digit using exactly the same format provided in the sample output below
4. Take a screenshot of each result to be submitted later

Sample Output for Part 2:

```
The valid credit card number is: 4614803459600174 and the newly computed check digit is: 4
```

Deliverables

readme.txt

So-called "read me" files are a common way for developers to leave high-level notes about their applications. Here's an example of a README file for the Apache Spark project. They usually contain details about required software versions, installation instructions, contact information, etc. For our purposes, your readme.txt file will be a way for you to describe the approach you took to complete the assignment so that, in the event you may not quite get your solution working correctly, we can still award credit based on what you were trying to do. Think of it as the verbalization of what your code does (or is supposed to do). Your readme.txt file should contain the following:

1. **Name**: Your name and JHED ID
2. **Module Info**: The Module name/number along with the title of the assignment and its due date
3. **Approach**: a detailed description of the approach you implemented to solving the assignment. Be as specific as possible. If you are sorting a list of 2D points in a plane, describe the class you used to represent a point, the data structures you used to store them, and the algorithm you used to sort them, for example. The more descriptive you are, the more credit we can award in the event your solution doesn't fully work.
4. **Known Bugs**: describe the areas, if any, where your code has any known bugs. If you're asked to write a function to do a computation but you know your function returns an incorrect result, this should be noted here. Please also state how you would go about fixing the bug. If your code produces results correctly you do not have to include this section.

Please submit your ***luhn_verify.py*** and luhn_generate***.py*** source code files along with a PDF file containing screenshots of your 3 outputs in a PDF called JHEDID_mod3.pdf (ex: jkovba3_mod1.pdf). Please do not ZIP your files together.

Recap:

1. readme.txt
2. luhn_verify.py
3. luhn_generate.py
4. A PDF containing the 3 screenshots of your outputs from Part 1 and Part 2

Please let us know if you have any questions via Teams or email!