Fundamentals of Programming

First Project

September 26th, 2014

Credit Card Numbers

Credit card numbers share a common numbering scheme according to the norm ISO / IEC 7812. The most commonly used credit card number scheme ranges from 12 to 19 digits, as follows:

- Six initial digits identify the credit card issuer (known as "Issuer Identification Number" or IIN);
- These digits are followed by a variable number of digits corresponding to the account identifier associated to the card;
- A final check digit calculated using the Luhn Algorithm.

A credit card number's first digit (known as "Major Industry Identifier" or MII) reprseents the category of the entity whom issued the card. The MII defines the following issuer categories:

1: Airlines;

2: Airlines, financial and other future industry assignments;

3: Travel and entertainment;

4: Banking and financial;

5: Banking and financial;

6: Merchandising and banking/ financial;

7: Petroleum and other future industry assignments;

8: Healthcare, telecommunications and other future industry assignments;

9: National attribution.

For example, American Express and Diners Club International are in the travel and entertainment category; VISA and MasterCard belong to the banking and financial sector.

The IIN (including the MII) identifies the organization that issued the card. Table 1 presents some IIN values.

Issuing Organization	Abbreviation	IIN initial digits	Number of digits
American Express	AE	34 e 37	15
Diners Club International	DCI	309, 36, 38 e 39	14
Discover Card	DC	65	16
Maestro	M	5018, 5020, 5038	13 ou 19
Master Card	MC	50 a 54 e 19	16
Visa Electron	VE	4026, 426, 4405, 4508	16
Visa	V	4024, 4532, 4556	13 ou 16

Table 1: IIN composition.

The rest of the number is assigned by the issuer. Most credit card numbers are validated using the Luhn Algorithm:[1]

1. Remove the number's last digit. The last digit is the check digit;
2. Invert the number's order;
3. Multiply every odd digit (positions 1, 3, 5, etc.) by 2 and subtract 9 to every resulting number that's greater than 9;
4. Compute the sum of every number, including the check digit;
5. The remainder of the sum divided by 10 must be 0 (i.e. sum must be a multiple of 10).

Exemplo:

Step																
Origial number	4	5	5	6	7	3	7	5	8	6	8	9	9	8	5	5
Remove the last digit	4	5	5	6	7	3	7	5	8	6	8	9	9	8	5	
Invert the digits	5	8	9	9	8	6	8	5	7	3	7	6	5	5	4	
Duplica ímpares	10	8	18	9	16	6	16	5	14	3	14	6	10	5	8	
Subtrai 9	1	8	9	9	7	6	7	5	5	3	5	6	1	5	8	
Compute the sum	90															
Sum mod 10	0															

1 Assignment

With this project, it is intended that a program is written using the Python programming language that performs two kinds of operations: (1) verify the correction of a credit card number, and (2) generate correct credit card numbers. To accomplish that:

1. You should write a program that receives a number corresopnding to a credit card, determine if this number is correct and, if it is, indicate the card's issuing organization and the its category (based on table 1).

   Your program should be invoked through the `check_cc` function which takes an integer corresponding to a possible credit card number as an argument and

[1]Patented in 1960 by Hans P. Luhn (U.S. Patent 2,950,048, *Computer for Verifying Numbers*).

returns a tuple containing the category and the issuing organization, if the number matches a credit card, or the string `'invalid card'` otherwise. For example:

```
>>> check_cc(38153682601755)
('Travel and entertainment', 'Diners Club International')
>>> check_cc(4532728243332223)
('Banking and financial', 'Visa')
>>> check_cc(4556223160581)
'invalid card'
>>> check_cc(5485060128076517)
('Banking and financial', 'Master Card')
>>> check_cc(6554615813812884)
'invalid card'
```

2. You must write a program which takes a string argument corresponding to the abbreviation of an issuing organization whose category is "banking and financial" and generates a valid credit card number issued by the given organization (based on table 1).

   The account identifier created by the issuing organization should be randomly generated using the `random()` function, located in the `random` module, which generates pseudo-random numbers in the interval [0, 1[. If for a given issuer there are multiple IIN numbers or different values for the credit card number's size, your program should choose one of them at random.

   Your program must be invoked through the `generate_cc_num` function which takes a string argument corresponding to the abbreviation of an issuing organization (see table 1). For example:

```
>>> generate_cc_num('MC')
1953081084203475
>>> generate_cc_num('DC')
6567518003213645
>>> generate_cc_num('V')
4532440306227148
>>> generate_cc_num('V')
4556245018079
```

2 Few suggestions

Taking into account the operations that will have to be performed on credit card numbers, these numbers should be converted to strings. To do so, use the built-in `str` function; for example, `str(5364)` returns the string `'5364'`.

2.1 Credit card number verification

To verify the validty of a credit card number, the following aspects must be verified:

1. The number must satisfy the Luhn Algorithm;
2. The number's prefix, i.e., its initial digits, must be one of those indicated in Table 1. For example, the number `8764532910871` isn't valid because an IIN starting with the digit `8` doesn't exist.
3. The length of the number must be one of the possible lengths for the given IIN. This information is represented in Table 1 as well. For example, the number `402687645329` isn't valid because the numbers starting with the prefix `4026` must be 16 digits long.

The Luhn Algorithm

In order to verify if a number satisfies the Luhn Algorithm, start by writing the function `compute_sum` which takes a string as its argument, representing a credit card number, without the check digit. The function returns the weighted sum of the `n` digits, calculated according to the Luhn Algorithm. For example,

```
>>> compute_sum('3248')
    18
```

Since the sum must be computed and one can't sum strings, use the `eval` function; this built-in funtion the integer number represented by a given numeric string. For example, `eval('8')` returns the integer `8`.

Using the `compute_sum` function, write the `luhn_verify` function which takes a string argument, representing a credit card number, and returns *true*, if the number satisfies the Luhn Algorithm, and *false* otherwise. This function doesn't verify if the number matches any issuing organization. For example,

```
>>> luhn_verify('4556245018079')
True
>>> luhn_verify('4556245018072')
False
```

Number's prefix

By analyzing Table 1, we conclude that the IIN prefixes that identify the issuing organizations can have different lengths; we can also conclude that some issuers have a single prefix (*Discover Card*, for example), and other issuers have multiple possible prefixes(*Visa Electron*, for example).

Start by writing the function `starts_with` which takes two strings `str1` and `str2`. The function returns *true* if `str1` starts with `str2`, and *false* otherwise. For example,

```
>>> starts_with('12345678', '123')
    True
>>> starts_with('12345678', '23')
    False
>>> starts_with('123', '12345678')
    False
```

Use this function to write the function `starts_with_one` which takes a string argument, `str`, and a string tuple argument, `strs`, and returns *true* only if `str` starts with any of the elements in the `strs` tuple. For example,

```
>>> starts_with_one('36238462919584', ('309',' 36', '38', '39'))
    True
>>> starts_with_one('36238462919584', ('34', '37'))
    False
```

Issuing Organization

Write the function `validate_iin` which takes in a string argument, representing a credit card's number. The function returns a string corresponding to the issuing organization, if one exists. Otherwise, it returns an empty string. This function only verifies the initial digits and the string's length. For example,

```
>>> validate_iin('4508654345231273')
    'Visa Electron'
>>> validate_iin('45086')
    ''
```

Note that the first example doesn't satisfy the Luhn Algorithm, and, consequently, isn't a valid number. That verification isn't performed by the `valiate_iin` function.

MII

Lastly, write the function `category` which takes a string argument, and returns a string corresponding to the category of the issuer corresponding to the string's first character. The behavior for when a category doesn't exist or is incorrect is undefined. For example,

```
>>> category('193')
    'Airlines'
>>> category('1')
    'Airlines'
```

Note: the usage of accented characters is ill-advised, both in strings or comments.

And now it's simple... just use the functions you've defined to write `check_cc`.

2.2 Generating a credit card number

In order to generate a credit card number belonging to a given issuer, one should firstly choose an issuing organization and the length of a number, base on Table 1. For example, se the issuer is "Maestro", the prefix could be either 5018, 5020 or 5038, and the length could be 13 or 19.

Secondly, digits between 0 and 9 should be appended (to the chosen prefix), up until the number reached the chosen length minus one. If in the previous example you would've chosen the prefix '5020' and the length 13, the number 5020726819227 could've been generated, with length 12.

Lastly, the check digit should be appended. To generate this digit, use the function `compute_sum`, passing it the generated string, and obtain the remainder of the sum divided by 10. If the remainder is 0, the digit should be 0. Otherwise, the check digit must be calculated by your program. For example, `compute_sum('30229065652764')` returns 53 and, consequently, the check digit should be 7. Hence, a complete randomly generated credit card number could be 302290656527647.

Write the function `check_digit` which takes a string argument, representing a credit card number, without the last digit, and returns the check digit to which it should be appended in order to obtain a valid credit card number. For example, `check_digit('30229065652764')` returns the string `'7'`, not the integer `7`.

Now, use the functions described above to write th function `generate_cc_num()`.

2.3 Note

It is important that your functions scrupulously follow the given specifications, in particular with regard to the arguments' and returning values' data types.

After writing each function, test it with different arguments until you're convinced it's correct. After written and tested, you should proceed with writing the following functions.

3 Grading

The execution's evaluation will be performed by Mooshak, an automatic project delivery system. There are plenty of tests configured in the system. The execution time of each test is limited, as well as the memory used in each one. Submissions are limited, only being able to submit at least 15 minutes after the previous one. Only a maximum of 10 submissions are allowed simultaneously, which means a submission could be refused if this limit is exceeded. In that case, try submitting at a later time.

The tests considered for evaluation purposes may or may not include the examples available in this statement, paired with a set of additional tests. The fact that a project completes the given examples successfully doesn't imply, of course, that it is completely correct, since the examples that were given aren't too exhaustive. It's every group's responsibility to guarantee that the fashioned code is correct.

No kind of information about the test cases used by the autmoatic evaluation system will be made available. The test files used in the project's evaluation will be available in the course's subject's page after the delivery date.

The project's grade will be based on the following aspects:

1. Correct execution (70%).

   This part of the evaluation is done resorting to an automatic evaluation program that suggests a grade according to the various aspects that were considered.

2. Readability, namely procedural abstraction, well-chosen names, comment quality (and not quantity) and functions' size (25%).

3. Programming style (5%).

4 Fulfillment conditions and deadlines

The 1st project's devliery will be made exclusively by electronic means. The project should be submitted through the Mooshak system, by **11:59 PM** of **October 24th, 2014**. Tardy submissions will not be accepted, whatever the excuse may be.

A single file with the .py extension should be submitted with all of the project's code. The file should contain a comment, on the first line, indicating the group's students' names and numbers, as well as the group number.

The source file shouldn't contain accented characters or any non-ASCII character. This includes comments and strings. Non-compliant programs will result in a score penalization of 3.

Two weeks before the deadline, instructions on how to submit code on Mooshak will be published in the course's subject's page. Only then will it be possible to submit by electronic means. By then, the credentials needed to access the system will be supplied to every group. Up until the due date, multiple submissions can be made at will, being the last submission used for evaluation purposes. Therefore, a careful verification of the last submission should be done, guaranteeing it's the version of the project that is pretended to be evaluated. No exceptions will be made.

There's the possibility of an oral discussion of the project and/or a demonstration of the project's functionality (will be decided individually).

Copies, or very identical projects, will be penalized with failure to the subject. The faculty will be the only judge of what is considered or not to copy a project.