Penn Engineering
ONLINE LEARNING

Assignment 4.6: Functions

Summary

In this graded assignment, you will write a Python function that implements various algorithms.

Learning Outcomes

In completing this assignment, you will:

- Implement a function in Python using parameters and return values
- Write code that operates on strings of characters and treats them as numbers

Description

Credit card companies and banks use built-in security measures when creating the account numbers on credit cards to make sure the card numbers follow certain rules (you didn't think they were random, did you?). This means that there are only certain valid credit card numbers, and validity can quickly be detected by using an algorithm that may involve adding up parts of the numbers or performing other checks.

In this activity, you will implement a function that determines whether or not a card number is valid, according to some simple algorithms. Note that these algorithms are purely made-up; don't try to use them to create fake credit card numbers! :-)

We will assume that the credit card number is a string consisting of 14 characters and is in the format ####-####-####, including the dashes, where '#' represents a digit between 0-9, so that there are 12 digits overall. We will revisit this assumption in an optional later activity.

In the space below, implement a function called "verify" that takes a single parameter called "number" and then checks the following rules:

1. The first digit must be a 4.
2. The fourth digit must be one greater than the fifth digit; keep in mind that these are separated by a dash since the format is ####-####-####.
3. The sum of all digits must be evenly divisible by 4.
4. If you treat the first two digits as a two-digit number, and the seventh and eighth digits as a two-digit number, their sum must be 100.

```
1   def verify(number) : # do not change this line!
2
3     # write your code here so that it verifies the card number
4
5     # be sure to indent your code!
6
7     return True # modify this line as needed
8
9   input = "5000-0000-0000" # change this as you test your function
10  output = verify(input) # invoke the method using a test input
11  print(output) # prints the output of the function
12  # do not remove this line!
```

The rules must be checked in this order, and if any of the rules are violated, the function should return the violated rule number, e.g., if the input is '4238-0679-9123', then the function should return 2, indicating that rule #2 was violated because although rule #1 was satisfied (the first digit is a 4), rule #2 was not, since the fourth digit (which is 8) is not one greater than the fifth (which is 0). If all rules are satisfied, then the function should return True.

Note that the card number is not actually a number, but is a **string** of characters. In Python, you can generally use a string the same way you would use a list, e.g., accessing individual characters using their 0-based index. *Hint: You will need to do this for checking all the rules.*

However, when you access a character using its 0-based index, Python will treat it as a character/letter and not a number, even if it's a digit, and you need to be careful about how you use it in mathematical operations. For instance, if you had the characters '1' and '2' and try to add them, Python would **concatenate** them and use them to form a longer string; in this case, you would get '12'. However, if you try to subtract, multiply, divide, etc. then Python will give you an error.

To convert a character/letter to a number, use the "int" function, e.g., "x = int('1')" will convert the character/letter '1' to the number 1 so that you can use it in mathematical operations. *Hint: you will need to do this for rules 2-4.*

You can test your code by changing the argument to the “verify” function (line 9 of the starter code) and trying different inputs to see if you are correctly detecting violations of the different rules. Click the “Run” button to run your program, which will invoke the “verify” function with the argument you specified, and then print out the output that is returned.

Hint: Here are some inputs you can try!

- "5000-0000-0000": violates rule #1
- "4000-0000-0000": passes rule #1, violates rule #2
- "4007-6000-0000": passes rules #1-2, violates rule #3
- "4037-6000-0000": passes rules #1-3, violates rule #4
- “4094-3460-2754”: passes all rules

Other hints:

- *As you write your program, implement and test the rules starting with the one that seems easiest. Once you feel that you have correctly implemented it and can detect violations, move on to the next one. That is, don’t try to implement all the rules all at once and then hope it works: approach this systematically (decompose the problem!), doing one rule at a time. But be sure that all the checks are done in the right order when you submit the program for grading.*
- *Although you will need to iterate over all characters in the string for rule #3 and can do so by treating the string as a list, you will need to treat some characters differently than others, since not all characters are digits. You may want to use the “range” function in the for-loop header, keeping in mind that range(x, y) generates a list from x to y-1, inclusive.*
- *In the same way that we can compare numbers, we can compare characters. Not only can we use the equals and not-equals comparison operators, but we can also use greater-than, less-than, etc. For instance, value >= ‘0’ will see if value is ‘0’, ‘1’, ‘2’, etc. but only if it is a character, e.g., ‘8’, but not if it is a number like 8.*
- *As in previous activities, don’t forget that you can use the “print” function to print out intermediate values as your code is performing operations so that you can see what is happening.*

While implementing your solution, you may run into some Python runtime errors related to the use of strings. Here are some of the common ones:

- **IndexError:** when accessing individual characters within a string using their indices, the same rules apply as with lists: if you have a string of *N* characters, then valid indices range from 0 to *N*-1, inclusive.
- **TypeError:** if the message mentions “unsupported operand types,” this probably means you are trying to perform mathematical operations on characters, not numbers. Be sure to convert characters to numbers using the “int” function.

- **TypeError:** a related error would include the message “[operation] not supported between instances” which means that you may be trying to compare a number to a character.
- **ValueError:** when using the “int” function, the character that you are providing must be a digit, e.g, ‘0’, ‘1’, ‘2’, etc. This error indicates that you might be trying to convert a non-numeric character to an int, particularly if the message indicates “invalid literal for int()”.

You may also encounter a **TypeError** if there is a problem related to the number of parameters in the function definition and the number of arguments provided when invoking the function; these would be indicated by an error message mentioning “positional arguments.” You should not change the first line of the function definition on line 1, and should only provide one argument when invoking the function at the bottom of your program.

After you have run your program using the “Run” button and believe that it is producing the correct output, accept the terms of the Coursera Honor Code and then click the “Submit Quiz” button below to submit this assignment and have it graded.

A few seconds after you submit the assignment, you will see the result at the top of the screen. If it reads “Congratulations! You passed!” then you are ready to proceed to the next lesson.

However, if it reads “Try again once you are ready.” then this means that the automatic grading program indicated that your program is not correct. You can find the error message from the grading program beneath your code. In that case, click the “Retake” button to go back to the quiz and try again.

If you believe that your code was correct, be sure to click the “Run” button before submitting and inspect the result of the “print” statement that is right before the “return” statement. This allows you to see the output that your code is producing before it is evaluated. If it looks right, but the automatic grading utility is still indicating that it is incorrect, then post a message on the discussion board to ask for assistance, but please do not post your code so that you do not reveal your solution to other learners.