COMP 1001 (Fall 2024): Practice Problem Set 3

Suggested Completion Date: Oct 20, 2024

Important Notes:

- Practice Problem Sets are not a part of the evaluation for the course.
- After the Suggested Completion Date, a rubric and solution outline will be posted on the course Brightspace shell so you can check your answers.
- Completing the Problem Sets will help you to gain hands-on experience with coding in Python, and will help you apply the concepts we have covered to solve and code solutions for the given problems. This will help you prepare for your tests and exam.
- While coding solutions for the problems given below, keep in mind that on the test/exam you will also be marked on the following:
  - *Efficient solution of the problem.* A given problem can be solved in a number of different ways, but the best solution is the one that is efficient; i.e., the one that uses the right concepts in a very productive way.
  - *Including sufficient descriptive comments in your program.* The person looking at your code should be able to understand how your code is solving the given problem even if the person reading your Python program does not know the Python language. In addition, the reader of your program should be able to understand what each variable represents.
  - *Labelling of input and output.* All input and output should have a descriptive label so that the reader of your program understands what input is expected and what output the program has generated.
  - *Program style* - consistent formatting and indentation of program statements, meaningful variable names (identifiers), and the use of constants (constant identifiers), where appropriate.

  Practicing these rules will build a good foundation for programming.
- This Problem Set is based on Chapter 5 of the textbook, without the graphics components. Please use only concepts from Chapters 1–5 of the textbook.

1. Write a Python program in a file called `processStrings.py` which, given a number of strings of digits, uses a `main` function to test the following functions:

   (a) `processStr` which, given a string of digits, computes and prints the sum and the product of the numbers in the string;

   (b) `getHighest` which returns the largest digit/number in the string;

   (c) `getHasRepeatDigit` which calls and returns the result from the function `getHighest` if the string has no repeated digits and zero otherwise.

   Sample input/output:

```
Enter a string of digits to process or 'quit' to stop: 3294751
The sum of the digits in 3294751 is 31
The product of the digits in 3294751 is 7560
The string has no repeated digits.
The highest digit in the given string is: 9
Enter a string of digits to process or 'quit' to stop: 454182
The sum of the digits in 454182 is 24
The product of the digits in 454182 is 1280
The string has repeated digits.
Enter a string of digits to process or 'quit' to stop: 832015
The sum of the digits in 832015 is 19
The product of the digits in 832015 is 0
The string has no repeated digits.
The highest digit in the given string is: 8
Enter a string of digits to process or 'quit' to stop: 22222
The sum of the digits in 22222 is 10
The product of the digits in 22222 is 32
The string has repeated digits.
Enter a string of digits to process or 'quit' to stop: quit
```

2. Write a Python program, in a file called `card_valid.py`, to implement the following algorithm for validating credit card numbers: (for illustration, consider the card number 4388576018402626):

   (a) The card number must be between 13 and 16 digits.

   (b) The card number must start with 4, 5, 6, or 37.

   (c) The card number must pass the following validity check:

      i. Double every second digit from right to left. If doubling of a digit results in a two-digit number, add the two digits to get a single-digit number.

      ii. Add all single-digit numbers from Step (i).

      `4+4+8+2+3+1+7+8 = 37`

      iii. Add all digits in the odd locations (counting from one) from right to left in the card number.

6+6+0+8+0+7+8+3 = 38

iv. Sum the results from Steps (ii) and (iii).

37+38 = 75

v. If the result from Step (iv) is divisible by 10, the card passes this validity check; otherwise it fails.

For example, the number 4388576018402626 is invalid, but the number 4388576018410707 is valid. Your program should accept a credit card number from the user **as an integer**, and output whether the number is valid or invalid. Use the following functions in your solution:

```
# Return True if the card number is valid
def isValid(number):

# Get the result from Step (b)
def sumDoubleEvenLocation(number):

# Return number if it is a single digit, otherwise, return
# the sum of the two digits in number
def getDigit(number):

# Return sum of odd place digits in number
def sumOddLocation(number):

# Return True if the d is a prefix for number,
# ie. appears at the beginning of the number
def isPrefix(number,d):

# Return the number of digits in d
def getSize(d):

# Return the first k digits from number.  If the number
# of digits in number is less than k, return number
def getPrefix(number,k):
```