CSCE 110: Programming I

Lab #7 (100 points)

Due: Sunday, October 16 by 11:59pm

1 Please make sure you understand the following.

For this assignment, you are only allowed to use what we have discussed during the first 7 weeks of class.

Please label your Python programs `q<num>.py`, where `<num>` is the question number. Take your time and make sure you understand everything in this lab before getting started. Also, make sure your programs match the output EXACTLY as given for each question.

2 Lab Questions

1. *Password validator.* Write a program (called q1.py) to check the validity of a password chosen by a user. To be considered valid, a password

   a) contains at least 1 letter between [A-Z],

   b) contains at least 1 letter between [a-z],

   c) contains at least 1 number between [0-9],

   d) contains at least 1 special character from [$#@],

   e) has a minimum length of 6 characters, and

   f) has a maximum length of 12 characters.

   Your program will consist of two user-defined functions: `validate(s)` and `main()`. The `validate()` function implements the validation procedure described above. The parameter (or input) to the function is a string s. If s fits the above criteria, print valid. Otherwise, print not valid. The `main()` function drives the program and has been written for you. **Your goal is to write the validate() function.**

   Example. The user input (which is handled in the provided main() function) is shown on lines with the > symbol. The code you write for the validate() function will provide the output for the lines that are not prefixed with the > symbol. On line 1, the user wants to know if the string `F1#` is a valid password (line 1). `F1#` is not a valid password (line 2). Next, the user wants to know of `ABd1234@1` is valid (line 3) and it is a valid password (line 4). The program continues in this manner, where the user enters a password and the output shows whether the password is valid or not (lines 5–16). Once the user is done checking password, quit exits the program (line 17).

```
 1  > F1#
 2  not valid
 3  > ABd1234@1
 4  valid
 5  > 2w3E*
 6  not valid
 7  > 2We3345
 8  not valid
 9  > aBD123BD@
10  valid
11  > ***ABCD***D%#a
12  not valid
13  > **ABCD*D%#a1
14  valid
15  > !23
16  not valid
17  % quit
```

2. *Reversing and adding numbers.* You will write a Python program (called q2.py) to implement the following procedure to produce palindromic numbers.

   a) Take any positive integer x. If x is a palindrome, stop. Otherwise, continue to step (b).

   b) Reverse the digits of x.

   c) Add the reverse to x.

   d) If the sum is a palindrome, then stop. Otherwise, let x represent the sum, and repeat steps (b) through (d).

   Steps (b) through (d) represent a single iteration. Below, are the number of iterations required for a few numbers to become palindromic using the reverse and add procedure.

   - 8 is a palindrome. It is palindromic after 0 iterations.
   - 56 becomes palindromic after 1 iteration: $56 + 65 = 121$.
   - 57 becomes palindromic after 2 iterations: $57 + 75 = 132$, $132 + 231 = 363$.
   - 59 becomes palindromic after 3 iterations: $59 + 95 = 154$, $154 + 451 = 605$, $605 + 506 = 1111$.
   - 89 takes an unusually large 24 iterations (the most of any number under 10,000 that is known to resolve into a palindrome) to reach the palindrome 8813200023188.
   - For 196, it is not known if the above procedure leads to a palindromic number.

Your program will consist of two user-defined functions: `reverse_add(low, high)` and `main()`. The `reverse_add()` function implements the reverse and add numbers procedure described above for all integers in the range low to high. For example, if low is 10 and high is 50, then the function would run the reverse and add procedure on the numbers 10, 11, . . . , 49, and 50. Or, the user could be interested in a single number such as 89. In this case, low and high are both 89.

For each number in the range of interest, your function will either output the number of steps required for it be palindromic or report that it does not lead to a palindrome within 100 steps. If the low and high range includes a single number (i.e., 190 to 190), then your program will show each step of the reverse and add procedure. See examples for more details.

The `main()` function drives the program. q2.py has been provided, where the code for the main() function is provided for you. **You will provide the code for the reverse_add() function.**

Example. The user input (which is handled in the provided main() function) is shown on lines with the > symbol. The code you write for the reverse_add() function will provide the output for the lines that are <u>not</u> prefixed with the > symbol. On line 1, the user wants to know if the integers between 10 and 15 lead to a palindrome (line 1). All of the numbers in this range lead to a palindrome in 1 step or less (lines 2–7). Next, the user wants to know if the integers in the range 84 to 91 lead to palindromes (line 8). The resulting output is shown in lines 9 to 16.

Line 17 shows that the user is only interested in a single number, 190. In other words, the range is all numbers between 190 and 190. When the range between the low and high integers is 1, then the output shows each step of the reverse and add procedure (lines 18–24) and shows a summary of the result (lines 26). For integers in the range 195 to 200 (line 27), every number except for 196 leads to a palindrome (lines 28–33). The user enters 76 (line 34) and the entire calculation is shown (lines 35–38). Finally, the user then quits the program at line 39.

```
 1  > 10 15
 2  10: PAL (1 step)
 3  11: PAL (0 steps)
 4  12: PAL (1 step)
 5  13: PAL (1 step)
 6  14: PAL (1 step)
 7  15: PAL (1 step)
 8  > 84 91
 9  84: PAL (2 steps)
10  85: PAL (2 steps)
11  86: PAL (3 steps)
12  87: PAL (4 steps)
13  88: PAL (0 steps)
14  89: PAL (24 steps)
15  90: PAL (1 step)
```

```
91: PAL (2 steps)
> 190
1. 190 + 091 = 281
2. 281 + 182 = 463
3. 463 + 364 = 827
4. 827 + 728 = 1555
5. 1555 + 5551 = 7106
6. 7106 + 6017 = 13123
7. 13123 + 32131 = 45254

190: PAL (7 steps)
> 195 200
195: PAL (4 steps)
196: NOT PAL (100 steps)
197: PAL (7 steps)
198: PAL (5 steps)
199: PAL (3 steps)
200: PAL (1 step)
> 76
1. 76 + 67 = 143
2. 143 + 341 = 484

76: PAL (2 steps)
> quit
```

3. *Cows and Bulls.* Write a program (called q3.py) that plays the Cows and Bulls game. The game works as follows. Randomly generate a 3-digit number, where every digit between 0 and 9 is unique. Ask the user to guess a 3-digit number. For every digit that the user guessed correctly in the correct place, they have a "cow". For every digit the user guessed correctly in the wrong place is a "bull." Every time the user makes a guess, report the number of "cows" and "bulls". Once the user guesses the correct number, the game is over.

   q3.py has been provided for you. It contains two user-defined functions: main() and welcome(). The `main()` function drives the program. The `welcome()` function prints a welcome message to the screen. The code for the `welcome()` function has been provided for you. **Your goal is to write the remaining part of the main() function.**

   Programming tips. When generating a 3-digit random number, make sure that every digit is unique. The best way to do this to keep generating a random number until the digits are unique.

   Note that since the program uses random number, your program will not run in the same manner as the following examples unless the same random numbers are generated.

   Example #1. Once the welcome message printed (lines 1–3), the user initially guesses 123 (line 5). The computer reports 0 cows and 0 bulls (line 6). The user then guesses 456 at the prompt (line 8), and the computer reports 1 cows, 2 bulls (line 9). The user guesses 465 for the third guess (line 11). The computer reports 0 cows, 3 bulls (line 12). The user keeps playing the game until the number is guessed correctly (lines 14–18). A congratulations message is printed (line 20) along with the secret number and number of guesses (line 21).

```
 1  -----------------------------
 2  | Welcome to Cows and Bulls |
 3  -----------------------------
 4
 5  Guess #1: 012
 6  0 cows, 0 bulls
 7
 8  Guess #2: 345
 9  0 cows, 2 bulls
10
11  Guess #3: 465
12  0 cows, 3 bulls
13
14  Guess #4: 546
15  0 cows, 3 bulls
16
17  Guess #5: 654
18  3 cows, 0 bulls
19
20  You got it!
21  It took you 5 guesses to guess the secret number 654.
```

Example #2. This example follows similarly to Example #1.

```
 1  -----------------------------
 2  | Welcome to Cows and Bulls |
 3  -----------------------------
 4
 5  Guess #1: 123
 6  0 cows, 1 bulls
 7
 8  Guess #2: 234
 9  0 cows, 0 bulls
10
11  Guess #3: 356
12  0 cows, 0 bulls
13
14  Guess #4: 789
15  1 cows, 1 bulls
16
17  Guess #5: 781
18  1 cows, 2 bulls
```

```

Guess #6: 718
3 cows, 0 bulls

You got it!
It took you 6 guesses to guess the secret number 718.
```