SY201 Name(s): ____________________

Alpha(s): ____________________

Assignment Type:	Lab	Collaboration Policy:	Default
Assignment Title:	Lab 5 Directions	Submit Project Name:	lab5
Electronic submission due: _________ Paper submission due: NO PAPER SUBMISSION Submission instructions: http://courses.cyber.usna.edu/SY201/calendar.php?load=policy			

1. Assignment Overview

In this assignment you will build a program with multiple pieces of functionality, all of which are focused on passwords.

2. Background Research

   a. Like most programming languages Python allows programmers to reuse code, i.e. incorporate source code that other developers wrote into new programs. Python provides this functionality by importing modules; module(s) is the term Python uses, other languages call them libraries. In Python the import word is reserved to import a Python module. Enter these commands into the Python interpreter and read the results.

      i. import hashlib # import the hashing library module
      ii. help(hashlib) # help information for the hash library module

   b. Importing modules is especially useful when dealing with cryptographic operations as any mistakes made in writing your own crypto can disable any cryptographic security benefit. But while the mathematics behind cryptography is well beyond the scope of SY201, you will see it later in the major.

   c. Because the hashlib module is extensive, we will give an example of how you will use it in this lab:

```
hashValue = hashlib.md5( bytes( "spam", "ASCII" ) ).hexdigest()
```

   This line of code computes the hexadecimal representation of the hash of the string *spam* and stores it in the variable hashValue. It uses the hash function md5 but other hash functions could be used (e.g., sha1 and sha256).

3. Specification - create a file lab5.py with the following functions and **includes comments adjacent to each function that describe the function's purpose and use**:

SY201 Name(s): ____________________

Alpha(s): ____________________

a. Write your alpha and section number in a comment at the top of your program

b. Write any sources of discussion/collaboration in a comment at the top of your program

c. Write a function named verify_password_requirements that:
   i. Takes one argument: (1) a string (a possible password)
   ii. OPTION: consider creating additional functions to break this problem into smaller pieces
   iii. Returns True or False depending on whether the password meets the below DoD requirements for a standard non-privileged user network account:
      - Must be at least 14 characters
      - At least two uppercase characters [A-Z]
      - At least two lowercase characters [a-z]
      - At least two numerical digits [0-9]
      - At least two special characters. For this lab, special characters are limited to:
        !@#$%^&*()-_=+

d. Write a function named determine_hash_function that:
   i. Takes one argument: (1) a string (a hash value)
   ii. Returns a string ("MD5", "SHA1", or "SHA256", depending on the hash function that was used to generate the hash value/digest)
   iii. HINT: Look at the characteristics of the hash values generated in lab4

e. Write a function named crack_password_no_stretching that:
   i. Takes two arguments: (1) a string (a hashed and salted password) and (2) a string (a salt)
   ii. Returns a string (the password corresponding to the two arguments)
   iii. The password will be made from the same character set as defined in (c) and not longer than three characters
   iv. HINT: your compute_hash function from lab4 may be helpful...

f. Write a function named determine_rounds that:
   i. Takes three arguments: (1) a string (a password), (2) a string (a salt), and (3) a string (a hashed, salted, and stretched password)
   ii. Returns an integer (the number of hashing rounds performed to process the first two arguments into the third argument)

g. Write a function named crack_password_with_stretching that:
   i. Takes three arguments: (1) a string (a hashed, salted, and stretched password), (2) a string (a salt), and (3) an integer (number of hash rounds)
   ii. Returns a string (the password corresponding to the arguments)
   iii. The password will be made from the same character set as defined in (c) and not longer than three characters
   iv. HINT: your compute_hash function from lab4 may be helpful...

SY201 Name(s): ____________________

Alpha(s): ____________________

h. Write a function named main that (in this order):
   i. Takes no arguments
   ii. Precisely prints out the following:

Options
(1) Check Password
(2) Crack Password
(3) Quit

   iii. Prompts the user for a numerical selection using "Selection: "
   iv. If the user chooses option three (user input: 3), your program should ask for no additional input and generate no additional output
   v. If the user chooses option one (user input: 1), your program should:
      1. Prompt the user for a password to check using "Password: "
      2. Print out "Valid Password" or "Invalid Password" based upon whether the password meets the listed requirements above
   vi. If the user chooses an option two (user input: 2), your program should:
      1. Precisely print out the following:

Options
(1) No Stretching
(2) Stretching

      2. Prompts the user for a numerical selection using "Selection: "
      3. If the user chooses option 2 (user input: 2), prompt the user for a number of rounds using "Rounds: "
      4. Prompt the user for a hash value using "Hash Value: "
      5. Prompt the user for a salt using "Salt: "
      6. Crack the password
      7. Print out the result as "Hash Value: [HASH_VALUE] - Password: [CRACKED_PASSWORD]"

i. EXTRA CREDIT (5 points): Write a function crack_password_unknown _stretching that:
   i. Takes three arguments: (1) a string (a hashed, salted, and stretched password), (2) a string (a salt), and (2) an integer (<u>maximum</u> possible number of hash rounds)
   ii. Returns a string (the password corresponding to the arguments)
   iii. The password will be made from the same character set as defined in (c) and not longer than three characters
   iv. HINT: if you are having an issue with speed, think how your implementation approach in determine_rounds might help you
   v. Crack the following password given the constraints below
      - Hash value: d8dcc1c9d2944942fb972d3a76379038
      - Maximum password length: 3 characters
      - Salt value: $cY22
      - Max rounds of stretching: 100
   vi. Put the cracked password in a comment at the top of your lab5.py
      - NOTE: do NOT submit a password without the code that cracked it