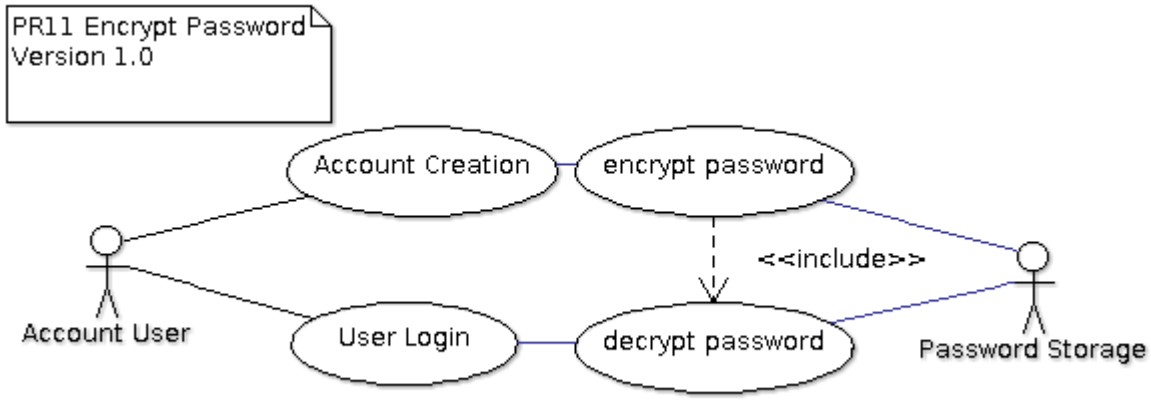
Problem Domain:	Login / Security / Encryption
Mission Summary:	Create a “Password Encoder”
Prerequisite:	Completion of “Python 1000” (available at Udemy.com)
Your Script Name:	PR11_PasswordEncoder.py
Solution Name:	PR11S_PasswordEncoder.py
Version:	1.0

Synopsis

Application security requires the validation of credentials. In order to avoid unauthorized account access, passwords should always be encrypted.



In this exercise, we will create a pair of functions that can be used to encrypt and decrypt a user’s password, or any other string.

Requirements

1) Create a script named PR11_PasswordEncoder.py
2) Define a function named “encrypt”
   A) INPUT:
      1) “User ID” (String)
      2) “Password” (String)
   B) OUTPUT:
      1) Success: Encrypted Password (String)
      2) Failure: None / NULL
3) Define a function named “decrypt”
   A) INPUT:
      1) “User ID” (String)
      2) “Encrypted Password” (String)
   B) OUTPUT:
      1) Success: Original Password

2) Failure: None / NULL
4) Cypher Strategy
A) ENCRYPTION LOGIC:
1) The content of the User ID will be iteratively added to the password
2) User ID iteration to “wrap around” whenever the length of the User ID is let than that of the Password
B) DECRYPTION LOGIC:
1) The content of the User ID will be iteratively subtracted from the password
2) User ID iteration to “wrap around” whenever the length of the User ID is let than that of the Password
5) Test Case
A) Verify Encryption / Decryption of:
B) Quotations are used in the above so as to express the requirement to include spaces as part of certain test patterns
C) Additional test cases should also be provided
6) Automated Testing Support:
A) Error messages should be of the format
1) “Error: x”
2) Where “x” is the error message
B) The testing success message should be of the format
1) “Success: x”
2) Where “x” can be any message (testing date time, etc.) desired
7) BONUS
A) Update User ID / Password exercises to use the encryption / decryption routines

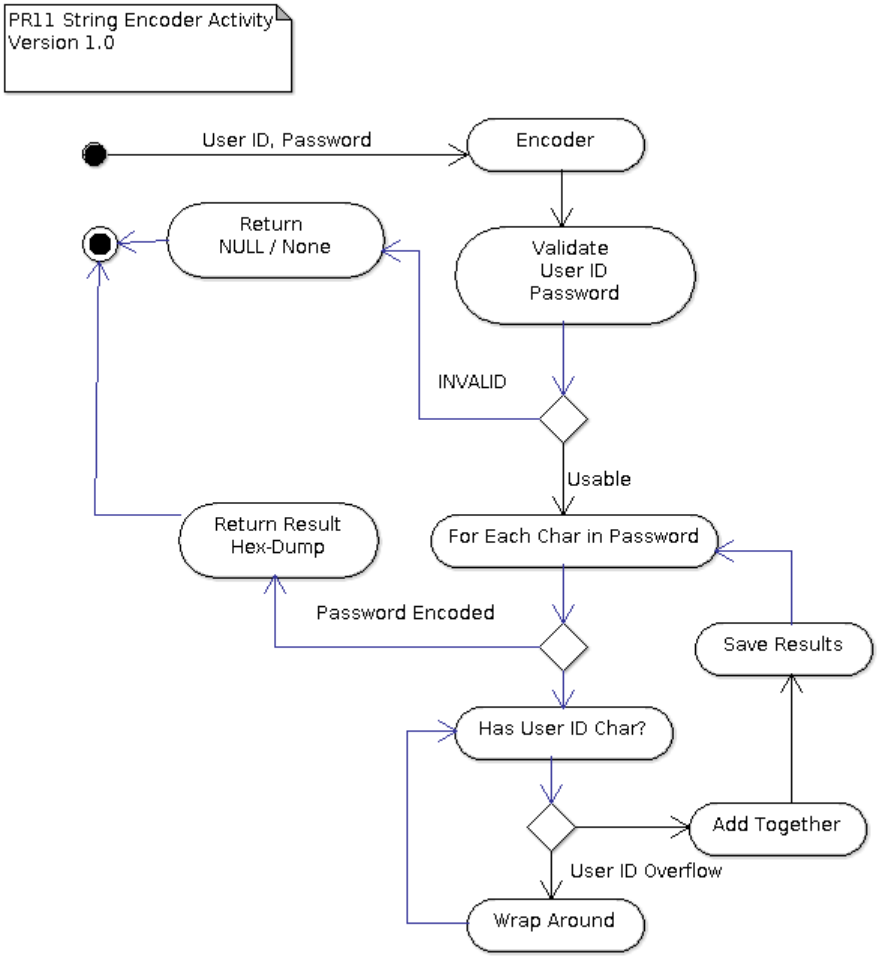
Developer Notes

1) The key requirement is to be able to mathematically add a single character offset from the user ID string, to a single character offset from the user’s password
2) The encryption strategy therefore requires us to know how to mathematical add an integral representation of any two characters together
3) In order to add two characters together:
A) We will first need to determine the integral value for a character
1) We can use Python’s built-in **ord()** function
2) Python’s ord() converts any single character to its integral / ordinal representation
B) To reverse the operation (i.e. turn an integer into a single character)
1) Requires the use of Python’s **char()** built-in function
2) Python’s char() function will convert an integer, to a single character
C) Once we can convert between any character’s string to its integral representation, we can combine the two by merely adding those character’s integral values together
4) Once combined together, the next challenge is to convert the additive-integer product to a hexadecimal string
A) We can use Python’s build-in **hex()** function to hex-dump any integer
B) To reverse the operation (i.e. convert a hexadecimal string representation to an integer) we can use **int(str, base)**, where “str” is the results returned from hex(), and “base” is the option base for the conversion – in this case, the number **16**.

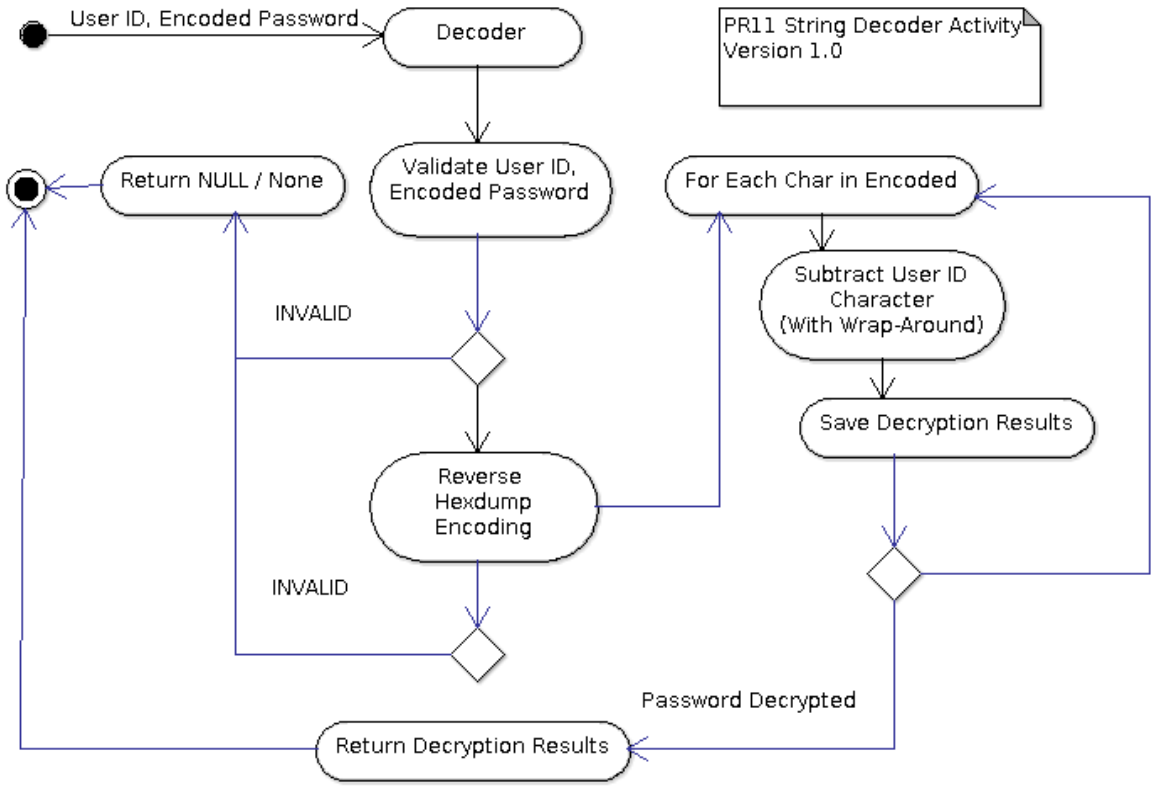
Related Diagrams

The following is a graphical requirement overview.

1.) The main Activity Diagram depicts a graphical summary of the key encryption requirements, as well as a reasonably demonstrative operational overview:



2.) The decryption routine complements the string / password encryption process:



(document ends)