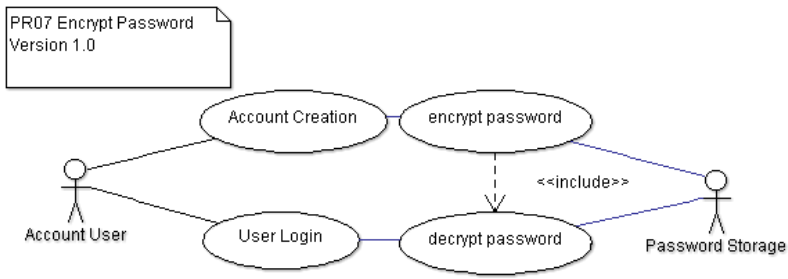
1000.07: Password Encryption

Problem Domain:	Login / Security / Encryption
Mission Summary:	Create a "Password Encoder"
Prerequisite:	Completion of "Python 1000"
Your Script Name:	PR07_PasswordEncoder.py
Solution Name:	PR07S_PasswordEncoder.py
Version:	1.0

Synopsis

Application security requires the validation of credentials. 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.

Cypher Strategy

1) ENCRYPTION LOGIC:
   a) The content of the User ID will be iteratively added to the password
   b) User ID iteration must "wrap around" whenever the length

of the User ID is less than that of the Password

2) DECRYPTION LOGIC:
   a) The content of the User ID will be iteratively subtracted from the password
   b) User ID iteration must "wrap around" whenever the length of the User ID is less than that of the Password

Requirements

1) Create a script named PR07_PasswordEncoder.py
2) Define a function named "encrypt"
3) INPUT:
   a) "User ID" (String)
   b) "Password" (String)
4) OUTPUT:
   a) Success: Encrypted Password (String)
   b) Failure: None / NULL
5) Define a function named "decrypt"
6) INPUT:
   a) "User ID" (String)
   b) "Encrypted Password" (String)
7) OUTPUT:
   a) Success: Original Password
   b) Failure: None / NULL
8) Test Case
   a) Verify Encryption / Decryption of a quotation
   b) Quotations may be used to test the above so as to test the inclusion of spaces, as well as punctuation (etc.)
9) Additional test cases should also be provided
10) Automated Testing Support:
   a) Error messages should be of the format "Error: x"
   b) Where "x" is the error message
11) The testing success message should be "Success: x"
   a) Where "x" can be any message (testing date time, etc.) desired

12) BONUS
    a) Update any User ID / Password exercises to use the encryption / decryption routines

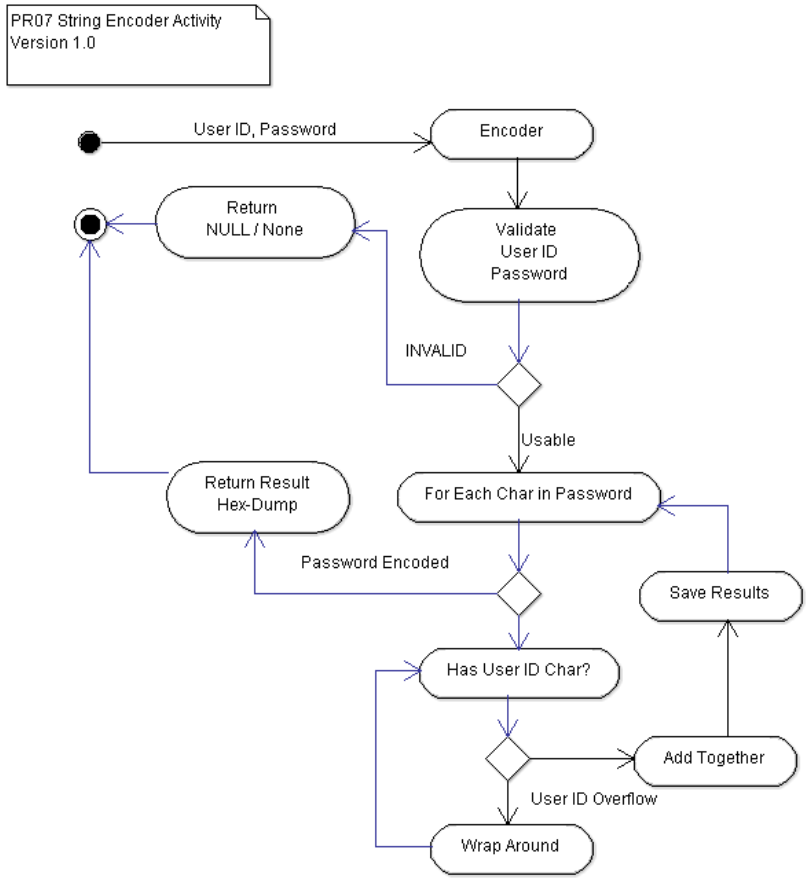
Developer Notes

1) The key requirement is to be able to mathematically add a single character from the User ID string, to a single character from the user's password
2) The encryption strategy therefore requires us to know how to mathematically 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
    b) We can use Python's built-in **ord()** function
    c) Python's ord() converts any single character to its integral / ordinal representation
4) To reverse the operation (i.e. turn an integer into a single character)
    a) Requires the use of Python's **chr()** built-in function
    b) Python's chr() function will convert an integer, to a single character
5) 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
6) Once combined together, the next challenge is to convert the additive-integer product to a hexadecimal string
7) We can use Python's build-in **hex()** function to hex-dump any integer
8) 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**.

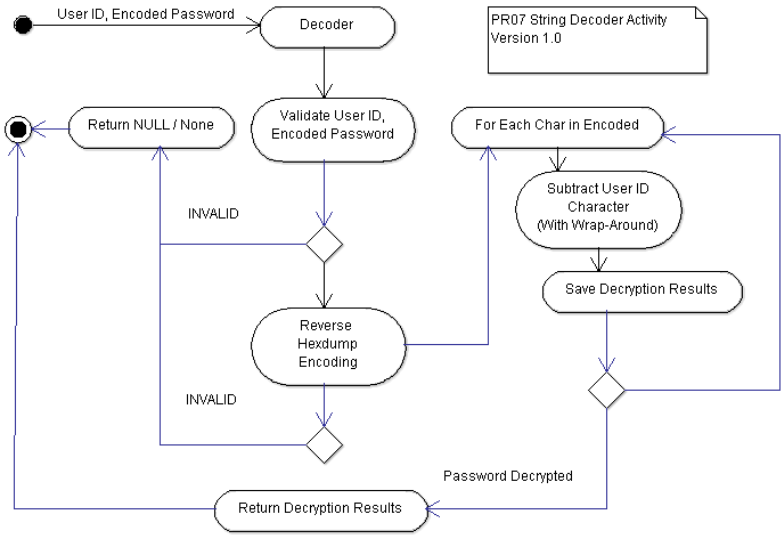
Related Diagrams

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:


User ID, Encoded Password
Decoder
PR07 String Decoder Activity
Version 1.0
Return NULL / None
Validate User ID,
Encoded Password
For Each Char in Encoded
Subtract User ID
Character
(With Wrap-Around)
INVALID
Save Decryption Results
Reverse
Hexdump
Encoding
INVALID
Password Decrypted
Return Decryption Results